Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет: Инженерно-экономический

Кафедра: Экономической информатики

Дисциплина: Средства и технологии анализа и разработки информационных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

АВТОМАТИЗИРОВАННАЯ СИСТЕМА УПРАВЛЕНИЯ РАБОТЫ ТУРФИРМЫ

БГУИР КП 1-40 01 02-02 06 ПЗ

Выполнил студент группы 372301

Иванова Л. В.

Проверил ассистент кафедры ЭИ

Ярмольчик В. В.

Минск 2016

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc437796209)

[1 Описание работы системы автоматизации розничного бизнеса банка 8](#_Toc437796210)

[1.1 Описание предметной области 8](#_Toc437796211)

[1.2 Принципы расчета в сфере розничного бизнеса банка 10](#_Toc437796212)

[2 Постановка задачи и обзор методов ее решения 15](#_Toc437796213)

[2.1 Анализ разрабатываемой системы. Постановка задачи на основе анализа 15](#_Toc437796214)

[2.2 Язык JAVA для разработки клиент-серверного приложения 16](#_Toc437796215)

[2.3 Сетевое программирование 17](#_Toc437796216)

[2.4 Сокетные соединения по протоколу TCP/IP 18](#_Toc437796217)

[2.5 Паттерн проектирования MVC 18](#_Toc437796218)

[2.6 Паттерн проектирования Адаптер 20](#_Toc437796219)

[3 Модели представления системы и их описание 22](#_Toc437796220)

[3.1 Диаграмма развертывания 22](#_Toc437796221)

[3.2 Диаграмма состояний 23](#_Toc437796222)

[3.3 Диаграммы последовательностей 24](#_Toc437796223)

[3.4 Диаграмма классов 27](#_Toc437796224)

[4 Функциональное моделирование на основе стандарта IDEF0 29](#_Toc437796225)

[5 Информационная модель системы и ее описание 32](#_Toc437796226)

[5.1 Краткие теоретические сведения 32](#_Toc437796227)

[5.2 Информационная модель розничного бизнеса банка 33](#_Toc437796228)

[5.3 Нормализация информационной модели 34](#_Toc437796229)

[6 Описание алгоритмов реализующих бизнес-логику серверной части проектируемой системы 36](#_Toc437796230)

[6.1 Алгоритм оформления поступивших заявлений 36](#_Toc437796231)

[6.2 Алгоритм закрытия кредитов 38](#_Toc437796232)

[6.3 Алгоритм закрытия депозитов 38](#_Toc437796233)

[6.4 Обобщенный алгоритм работы серверной части системы 40](#_Toc437796234)

[7 Руководство пользователя 42](#_Toc437796235)

[8 Результаты тестирования разработанной системы 57](#_Toc437796236)

[Заключение 62](#_Toc437796237)

[Список использованной литературы 63](#_Toc437796246)

[Приложение А (обязательное) Функциональная модель 64](#_Toc437796247)

[Приложение Б (обязательное) Листинг основных элементов программы 68](#_Toc437796248)

[Приложение В (обязательное) Листинг скрипта генерации базы данных 82](#_Toc437796249)

**ВВЕДЕНИЕ**

Понятия туризм и информация неразделимы. Это даёт основание утверждать, что каждый сотрудник, работающий в туристской отрасли, должен уметь работать с информацией, включая её сбор, обработку, и на этой основе принимать решения. Значительный объем информации, работающий в сфере туристского бизнеса, затрудняет ход производительного процесса. В связи с этим возникают задачи автоматизации производства, когда возможностей существующих коммуникаций становится недостаточно для эффективной работы. Таким образом, внедрение автоматизированных систем управления, основанных на использовании современных информационных технологий, крайне необходимо. Это позволяет туристской организации расширить диапазон услуг, предоставляемых клиенту, повысить оперативность работы и снизить издержки, более эффективно взаимодействовать с партнёрами, использовать мультимедийные рекламно-информационные продукты, а также получить доступ к обширным информационным ресурсам.

Современные компьютерные технологии активно внедряются в сферу туристского бизнеса, и их применение становится неотъемлемым условием повышения конкурентоспособности любого туристского предприятия. Индустрия туризма позволяет использовать все многообразие компьютерных технологий, начиная от специализированных программных продуктов управления отдельной туристской фирмой до применения глобальных компьютерных систем.

Специализированные автоматизированные системы управления дают возможность работать в режиме, предназначенном для формирования пакетов услуг и программ туров, готовых к реализации, калькуляции их стоимости, расчета прибыли. С помощью автоматизированных систем можно вести базы данных партнеров, что дает возможность организовать с ними персональную работу. С помощью комплексной системы автоматизации можно корректировать стратегию продаж. Использование специализированного программного обеспечение экономит время персонала, упрощает работу по оформлению туристов, обеспечивает печать всех необходимых документов, уменьшает вероятность ошибок в работе персонала.

Цель данного курсового проекта: сократить непроизводительные трудозатраты работников туристических фир, за счет частичной автоматизации процесса предоставления услуг; привлечение новых клиентов, за счет разработки удобной и дружественной инфраструктуры, с помощью которой клиентам станет приятнее пользоваться услугами.

Для достижения поставленной цели следует выделить основные задачи:

* изучить основные принципы и особенности предоставления туристических услуг клиентам;
* исследовать процессы бронирования и оформления путевок, определить пути их улучшения путем автоматизированной системы
* разработать модель, которая полно описывает функционирование системы, посредством различных наглядных диаграмм;
* определить методы и технологии разработки программного продукта;
* разработка приложения в соответствии с ранее составленной моделью;
* тестирование разработанной системы в реальной среде, анализ полученных данных и оптимизация;

# **1 ОПИСАНИЕ РАБОТЫ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ УПРАВЛЕНИЯ РАБОТЫ ТУРФИРМЫ**

## **1.1 Использование информационных технологий в туристической отрасли**

Международный и внутренний туризм представляет собой мощную отрасль по торговле услугами.

Основу туристической индустрии составляют фирмы туроператоры и турагенты, занимающиеся туристическими поездками, продажей их в виде путевок и туров; предоставляющие услуги по размещению и питанию туристов (гостиницы, кемпинги и др.), их передвижению по стране, а также органы управления, информации, рекламы по исследованию туризма и подготовке для него кадров, предприятия по производству и продаже товаров туристского спроса. На туризм работают и другие отрасли, для которых обслуживание туристов не является основным видом деятельности (предприятия культуры, торговли и др.).

Туризм – информационно насыщенная деятельность. Существует немного других отраслей, в которых сбор, обработка, применение и передача информации были бы настолько же важны для ежедневного функционирования, как в туристической индустрии. Услуга в туризме не может быть выставлена и рассмотрена в пункте продажи, как потребительские или производственные товары. Ее обычно покупают заранее и вдали от места потребления. Таким образом, туризм на рынке почти полностью зависит от изображений, описаний, средств коммуникаций и передачи информации.

Турагент – физическое или юридическое лицо, выступающее посредником по продаже сформированных туроператором туров. Туроператор – туристическая организация, занимающаяся комплектацией туров.

Система информационных технологий, используемых в туризме, состоит из компьютерной системы резервирования, системы проведения телеконференций, видеосистем, компьютеров, информационных систем управления, электронных информационных систем авиалиний электронной пересылки денег, телефонных сетей, подвижных средств сообщения и т.д. При этом необходимо отметить, что эта система технологий развертывается не турагентами, гостиницами или авиакомпаниями каждым в отдельности, а всеми ими. Более того, использование каждым сегментом туризма системы информационных технологий имеет значение для всех остальных частей. Например, системы внутреннего управления гостиницей могут быть связаны с компьютерными глобальными сетями, которые обеспечивают, в свою очередь, основу для связи с гостиничными системами резервирования, которые, уже в обратном направлении, могут быть доступны турфирмам через их компьютеры.

Следовательно, мы имеем дело с интегрированной системой информационных технологий, которая распространяется в туризме.

Кроме того, отдельные компоненты туристической отрасли тесно взаимосвязаны друг с другом — ведь многие турпроизводители вертикально или горизонтально вовлечены в деятельность друг друга. Все это позволяет рассматривать туризм как высокоинтегрированную услугу, что делает его еще более восприимчивым для применения информационных технологий в организации и управлении.

Не секрет, что, когда клиент туристического агентства, приобретая тур, получает хорошо подготовленные документы и полную информацию о туре, он более комфортно себя чувствует и больше доверяет выбранному агентству. Также и турфирма, затратив немалые средства на привлечение данного клиента к себе, сохраняя полную информацию о нем, может в дальнейшем использовать ее для повторного привлечения старого клиента к себе. Известно, что затраты в этом случае в четыре раза ниже первоначальных.

Есть возможность сразу поставить единую систему учета, как оперативного, так и финансового, причем в двух разрезах - управленческого и бухгалтерского. С использованием гибкого разделения прав доступа и видимости информации менеджерам доступна только их часть системы, а бухгалтерам - их часть. Естественно, руководители компании видят всю информацию и могут наблюдать за работой менеджеров со своего компьютера. Так как оперативный ввод информации ведется в единой системе, то видно, кто и как работает: количество введенных заявок, подтвержденных, оплаченных клиентом, оплаченных поставщику, закрытых, доход по каждой заявке. Через отчеты системы руководитель видит эффективность работы каждого менеджера и подразделения. Детализацию можно установить по каждому туристу и каждой составляющей тура. По любому клиенту можно увидеть историю его обращений и оплат.

В единой системе, менеджеры могут тут же увидеть, оплачена их заявка или нет, сумму оплаты. Бухгалтерия сразу же получает исходную информацию и не тратит время на повторный ввод одних и тех же данных. Вся информация разделена по правилам доступа, и лишнюю информацию менеджер не увидит. Соответственно, нет необходимости в разделении баз данных, что неминуемо приводит к ошибкам ввода, неоперативности работы и вытекающих из этого проблем.

После того как решены основные задачи систематизированного управления турфирмой – управление работой с клиентами, с поставщиками и внутренней деятельности компании, можно приступать к расширению системы.

В конечном итоге вся необходимая информация в красивом виде выводится на печать. Клиентам будет выдаваться весь комплект необходимых документов, поставщикам – соответствующие отчеты, а для государства – печататься все отчеты установленной формы.

Автоматизация работы турагентств обычно сводится к оперативному учету заказанных туристами туров, оформленных платежей с клиентами и партнерами, печати всех необходимых документов.

Для автоматизации отдельного туристического агентства необходимо использовать программное обеспечение, основанное на применении базы данных, аккумулирующей все рабочие данные в едином хранилище. Это необходимо для получения отчетов о работе агентства в любой момент времени. Кроме базы данных необходимо также иметь программу с удобным пользовательским интерфейсом, приемлемой скоростью работы, конфигурированием параметров системы, относящихся к деятельности фирмы. К таким параметрам в первую очередь относятся текст и внешний вид печатных документов, формируемых на базе рабочих данных, а также возможность управления правами доступа к различным данным системы.

С автоматизацией сетевых агентств дело обстоит несколько сложнее. Сетевые агентства с юридической точки зрения могут иметь различные типы: одна фирма с несколькими офисами продаж, сеть отдельных агентств под общим логотипом (франчайзи): Руководство сетевых агентств периодически нуждается в оперативной отчетности о деятельности всей сети. Предоставить такую отчетность можно только при условии хранения всех данных в едином хранилище.

Техническая сторона автоматизации сетевых агентств должна в определенном смысле переплетаться с видом сети. Если отдельное агентство достаточно легко автоматизировать, подключив его к серверу единой базы данных центрального офиса через каналы интернета, то с франчайзи так поступить невозможно. Проблемы связи с сервером базы данных сразу же приведут к параличу работы отдельного агентства. В случае с франчайзи это нарушение работы одной компании из-за проблем другой компании. Решение возможно только с помощью применения локальной рабочей базы данных в отдельном офисе, что исключает зависимость работы агентства от работы канала передачи данных, с копированием в центральный офис всех введенных данных (канал передачи данных используется достаточно редко по сравнению с постоянным подключением – раз в час, раз в день, периодичность выбирается головной компанией). Но за преимуществами кроются и недостатки. Сервер центрального офиса хранит у себя единую базу данных, собранную от отдельных агентств. Центральный офис может осуществлять только пассивное наблюдение за работой агентства. Технически очень сложно реализовать активное поведение центрального офиса по отношению к дочернему.

Автоматизацию точек продаж одной компании гораздо лучше проводить с использованием центральной базы данных и каналов постоянной связи с сервером. Такое построение сети позволяет гораздо эффективнее управлять данными (заявками, платежами, правами доступа и т.д.), а отчетность о работе (если на это есть необходимые права доступа) получать в любом офисе.

## **1.2 Системы бронирования**

Бронирование – это предварительный заказ на предстоящее туристское обслуживание, предварительное закрепление объектов бронирования: мест (билетов, путевок), снаряжения, оборудования и т.д. за заказчиком (Крутик А.Б. Внедрение систем электронного бронирования и резервирования. // Эффективное развитие туристского предприятия. - 2007.). Бронирование осуществляется различными информационными системами, которые резервируют, т.е. создают запасы объектов бронирования, а затем из созданного резерва закрепляют их за потребителем.

В настоящее время в сфере реализаций туристских услуг активно используется компьютерная техника. Сейчас уже трудно представить, как могло производиться бронирование отелей, железнодорожных или авиабилетов различными тур агентами и компаниями с учетом всевозможных скидок при отсутствии систем компьютерного бронирования.

Высокая экономическая эффективность использования систем компьютерного бронирования побудила их владельцев бороться за влияние на рынке туристских услуг. Для того чтобы привлечь тур агентов, компании расширили сферу применения систем бронирования, включив в их программу широкий круг услуг (бронирование гостиничных мест, прокат автомобилей, страхование авиапассажиров, оформление билетов, заграничных паспортов и даже заказов на цветы). Кроме того, программы предоставляют тур агентам возможность компьютерной обработки документации и бухгалтерского учета.

Функции агентства при самостоятельном бронировании клиентами через Интернет заключаются в проверке корректности выполненных броней, оптимизации выбранных маршрутов, выписке и доставке авиабилетов.

Итак, каждая компьютерная система бронирования создаёт свои собственные компьютерные программы, которые выполняют следующие функции:

* позволяют проводить поиск оптимальных ценовых вариантов туруслуги;
* позволяют работать по различным тарифам, в том числе, по конфиденциальным;
* позволяют турагенству автоматически распечатывать на матричном принтере билеты, предназначенные для ручной выписки, используя данные о бронировании;
* позволяют турагентствам использовать программу автоматического составления отчетов;
* позволяют вести различные виды статистики; предоставляют доступ к обширной справочной системе оперативной информации;
* позволяют контролировать все операции по взаиморасчетам с клиентами и поставщиками;
* позволяют создавать на сайте турагентства систему бронирования туруслуг для корпоративных и частных клиентов. Таким образом, посетитель сайта получает возможность самостоятельного выбора авиарейсов и отелей и их последующего бронирования (Осипова О. Я. Транспортное обслуживание. Обслуживание туристов. – М., 2004.).

Взаимодействие туроператора с компьютерными системами бронирования создает возможность:

* интегрироваться в мировой туристский рынок;
* оперативно управлять своей деятельностью;
* расширять рынок сбыта туристской продукции;
* планировать маркетинговую деятельность;
* предварительно формировать списки потребителей и заблаговременно обеспечивать загрузку;
* анализировать спрос на туристскую продукцию и многое другое.

## **1.3 Обзор современных программных и технических средств для решения подобных задач**

Выбор турфирмами программных продуктов и специфика их использования зависят от ряда факторов, в частности:

* от направления деятельности турфирмы, совокупности решаемых задач, исходной технологии, принятой на турфирме, суммарного объема продаж, финансового состояния турфирмы;
* от осведомленности руководства турфирмы о программном обеспечении, существующем на рынке информационных технологии управления в туризме, о его достоинствах и недостатках;
* от режима работы компьютеров (автономный или сетевой).

Существующие в настоящее время программы автоматизации работы туристских фирм можно условно разбить на три класса:

1  Программы автоматизации работ, которые обеспечивают внутреннюю деятельность турфирмы, без автоматизации поддержания внешних взаимосвязей. Предполагается создание локальной вычислительной сети, баз данных, которые располагаются на центральной мощной машине-сервере, куда обеспечивается сетевой доступ пользователей с их автоматизированных рабочих мест.

2  Программы автоматизации включают возможности программ 1-го класса, а также позволяют туроператору осуществлять модемную связь в пакетном режиме с другими турагентствами.

3  Программы автоматизации, базирующиеся на использовании глобальных телекоммуникационных сетей. В этом случае базы данных о турпродукте той или иной турфирмы размещаются в информационных узлах глобальной сети и для всех пользователей сети обеспечивают удаленный доступ в режиме online [44].

На диаграмме (рисунок 1.1) представлены ответы туристических агенств на вопрос: «Какие системы автоматизации турагенства вы знаете?».

По диаграмме можно сказать, что наиболее популярной системой является «САМО-турагент». Решение выбрать данное ПО является абсолютно обоснованным, так при анализе ряда подобных программ, «САМО-турагент» практически по всем рассматриваемым критериям получил положительный отзыв. Результаты анализа представлены на рисунках 1.2 и 1.3. [http://pro.tonkosti.ru/it-tehnologii\_v\_turizme/sistemy\_avtomatizatsii\_dlya\_turagentstv\_kakuyu\_vybrat-28792761]

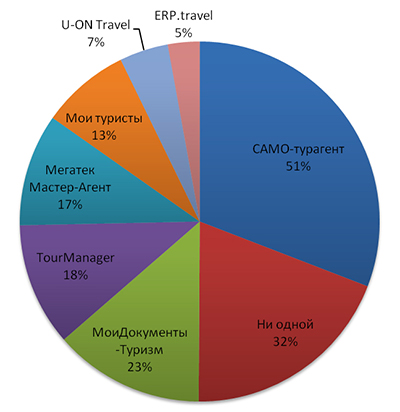
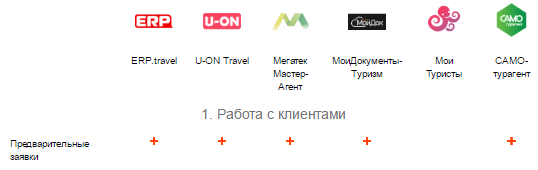


Рисунок 1.1 – Популярные системы автоматизации турфирмы





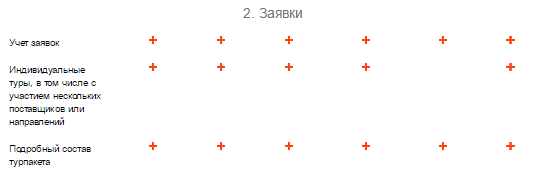


Рисунок 1.2 – Анализ систем по критериям





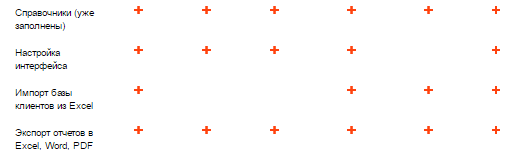




Рисунок 1.3 – Анализ систем по критериям (продолжение)

# **2 ОПИСАНИЕ ОСНОВНЫХ ПРОЦЕССОВ СИСТЕМЫ РАБОТЫ ТУРФИРМЫ**

Разработка автоматизированной системы состоит из нескольких этапов. Первоначально следует понять, какие именно функции должно выполнять разрабатываемое ПО, выделить основные компоненты программы и установить взаимосвязь между ними.

На рисунке 2.1 изображена диаграмма компонентов системы розничного бизнеса банка.



Рисунок 2.1 – Диаграмма компонентов разрабатываемой системы

Данная диаграмма показывает зависимости между основными компонентами системы, коими являются: клиент, служащий, программа клиента, сервер, база данных и услуга, состоящая из депозитов и кредитов.

Однако для более ясного представления работы именно программы составляются различные диаграммы вариантов использования. На рисунке 2.2 представлена система розничного бизнеса банка.

С помощью диаграммы можно определить основные функции, которая должна выполнять программа при работе с клиентом и служащим: принять заказы(заявления) от клиента и оформить их с последующим предоставлением услуг. Всеми операциями распоряжается служащий, а именно: он просматривает поступившие заявления, и посредством программы их оформляет.



Рисунок 2.2 – Диаграмма вариантов использования системы розничного бизнеса банка

Оптимальным решением для реализации выявленных функций и компонент программы является разработка клиент-серверного приложения с поддержанием многопоточности.

## **2.2 Язык JAVA для разработки клиент-серверного приложения**

Основное преимущество языка Java выражается в переносимости Java-приложений, т.е. способности работать на любых аппаратных платформах и операционных системах, поскольку все JVM, независимо от того, на какой платформе они работают, способны исполнять один и тот же байт-код [4].

Язык и платформа Java обладают великолепной масштабируемостью. Можно легко создавать приложения для устройств с ограниченными ресурсами, адаптируя программное обеспечение, изначально написанное для настольных компьютеров. При этом язык Java идеально подходит и для разработки серверных Web-приложений, при помощи которых пользователь может получать доступ к вычислительным ресурсам в Web. Возможность безопасного выполнения кода, загруженного через сеть, была изначально заложена в конструкцию Java, поэтому этот язык обеспечивает высокий уровень безопасности при работе через Интернет. Web-приложения работают в средах выполнения, называемых Web-контейнерами, которые предоставляют множество удобных сервисов, в том числе диспетчеризацию запросов, обеспечение безопасности и параллельной работы, управление жизненным циклом, а также доступ к таким API, как управление именами, транзакциями и электронная почта. На Java написан ряд серверов приложений, выполняющих функции Web-контейнеров для других компонентов Java, XML и Web-сервисов, которые взаимодействуют с базами данных и динамически формируют содержимое Web-страниц. Подобные серверы также предоставляют среду для развертывания корпоративных приложений и средства для управления транзакциями, кластеризации, обеспечения безопасности, связности и необходимого уровня доступности, производительности и масштабируемости [6].

## **2.3 Сетевое программирование**

Язык Java делает сетевое программирование простым благодаря наличию специальных средств и классов. Большинство этих классов находится в пакете java.net. Сетевые классы имеют методы для установки сетевых соединений передачи запросов и сообщений. Сервер должен поддерживать многопоточность, иначе он будет не в состоянии обрабатывать несколько соединений одновременно. В этом случае сервер содержит цикл, ожидающий нового клиентского соединения. Каждый раз, когда клиент просит соединения, сервер создает новый поток.

Сетевые приложения используют Internet-приложения, к которым относятся Web-браузер, e-mail, сетевые новости, передача файлов. Для создания таких приложений используются сокеты, порты, протоколы TCP/IP, UDP.

Приложения клиент/сервер используют компьютер, выполняющий специальную программу-сервер, которая обычно устанавливается на удаленном компьютере и предоставляет услуги другим программам-клиентам. Клиент – это программа, получающая услуги от сервера. Клиент устанавливает соединение с сервером и пересылает серверу запрос. Сервер осуществляет прослушивание клиентов, получает и выполняет запрос после установки соединения. Результат выполнения запроса может быть возвращен сервером клиенту. Запросы и сообщения представляют собой записи, структура которых определяется используемыми протоколами [5].

## **2.4 Сокетные соединения по протоколу TCP/IP**

Сокеты (сетевые разъёмы) – это логическое понятие, соответствующее разъёмам, к которым подключены сетевые компьютеры и через которые осуществляется двунаправленная поточная передача данных между компьютерами. Сокет определяется номером порта и IP-адресом. При этом IP-адрес используется для идентификации компьютера, номер порта – для идентификации процесса, работающего на компьютере. Когда одно приложение знает сокет другого, создается сокетное протоколо-ориентированное соединение по протоколу TCP/IP. Клиент пытается соединиться с сервером, инициализируя сокетное соединение. Сервер прослушивает сообщение и ждет, пока клиент не свяжется с ним. Первое сообщение, посылаемое клиентом на сервер, содержит сокет клиента. Сервер, в свою очередь, создает сокет, который будет использоваться для связи с клиентом, и посылает его клиенту с первым сообщением. После этого устанавливается коммуникационное соединение [7].

## **2.5 Паттерн проектирования MVC**

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться [9].

Для разработки более структурированного кода и приложения в целом можно использовать специально предназначенный для этого паттерн.

MVC (Model-View-Controller) — это фундаментальный паттерн, который нашел применение во многих технологиях, дал развитие новым технологиям и каждый день облегчает жизнь разработчикам (рисунок 2.3).

Впервые паттерн MVC появился в языке SmallTalk. Разработчики должны были придумать архитектурное решение, которое позволяло бы отделить графический интерфейс от бизнес-логики, а бизнес логику от данных. Таким образом, в классическом варианте, MVC состоит из трех частей, которые и дали ему название:

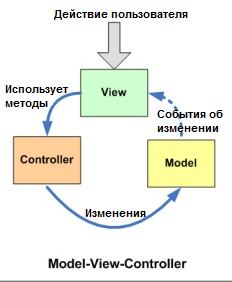


Рисунок 2.3 – Паттерн MVC

Модель.

Под Моделью, обычно понимается часть, содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. Достигается результат, позволяющий менять представление данных, то как они отображаются, не трогая саму Модель.

Модель обладает следующими признаками:

* модель — это бизнес-логика приложения;
* модель обладает знаниями о себе самой и не знает о контроллерах и представлениях;
* для некоторых проектов модель — это просто слой данных (DAO, база данных, XML-файл);
* для других проектов модель — это менеджер базы данных, набор объектов или просто логика приложения.

Представление (View).

В обязанности Представления входит отображение данных полученных от Модели. Однако, представление не может напрямую влиять на модель. Можно говорить, что представление обладает доступом «только на чтение» к данным.

Представление обладает следующими признаками:

* в представлении реализуется отображение данных, которые получаются от модели любым способом;
* в некоторых случаях, представление может иметь код, который реализует некоторую бизнес-логику;
* для некоторых проектов модель — это просто слой данных (DAO, база данных, XML-файл);

Основная идея этого паттерна в том, что и контроллер, и представление зависят от модели, но модель никак не зависит от этих двух компонент.

Признаки контроллера:

* контроллер определяет, какие представление должно быть отображено в данный момент;
* события представления могут повлиять только на контроллер, контроллер может повлиять на модель и определить другое представление;
* возможно несколько представлений только для одного контроллера;

Реализация:

Контроллер перехватывает событие извне и в соответствии с заложенной в него логикой, реагирует на это событие изменяя Модель, посредством вызова соответствующего метода. После изменения Модель использует событие о том, что она изменилась, и все подписанные на это события Представления, получив его, обращаются к Модели за обновленными данными, после чего их и отображают [8].

## **2.6 Паттерн проектирования Адаптер**

Адаптер преобразует ("адаптирует") один интерфейс в другой интерфейс (рисунок 2.4).

Может существовать множество причин для такого преобразования, например:

1 Класс А имеет сложный интерфейс, скорее "низкоуровневый". Для многих операций приходится делать несколько вызовов методов этого класса. Имеет смысл переработать этот интерфейс в более простой (в разрабатываемой системе удобно использовать при работе с базой данных).

2 Интерфейс класса А очень широк; но для определенных операций нам нужен более узкий, или специализированный интерфейс.

Например, один и тот же класс может использоваться клиентами (удаленно), но может также быть использован администратором сервера. В таком объект не должен "показывать" клиентам всю функциональность.

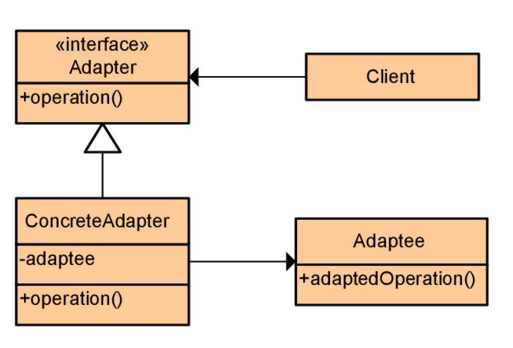


Рисунок 2.4 – Шаблон проектирования Адаптер

Есть 2 основных варианта реализации Адаптера: через наследование и через композицию [8].

# **3 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ И ИХ ОПИСАНИЕ**

## **3.1 Диаграмма развертывания**

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. Если создается простая программа, которая может выполняться локально на компьютере пользователя, не используя никаких распределенных устройств и сетевых ресурсов, то необходимости в разработке дополнительных диаграмм нет. Однако при создании корпоративных или распределенных приложений требуется визуализировать сетевую инфраструктуру программной системы.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются [10].

На рисунке 3.1 изображена диаграмма развертывания разрабатываемого приложения.



Рисунок 3.1 – Диаграмма развертывания системы

Данная диаграмма включает в себя два пользовательских узла и один серверный, а также один терминал, который связан с пользовательским узлом. У пользовательских узлов средой выполнения является «MS .NET Framework», в то время, как у серверной – «SQLServer». Среда выполнения «SQLServer» включает в себя компонент базы данных. Для взаимодействия с БД узел хранит файл с расширением .jar. Пользовательские узлы имеют исполняемые файлы с расширением .exe и архивы .jar, которые содержат в себе части программы. Основной принцип взаимодействия узлов: пользовательские узлы – Workman(Admin)Station и ClientWorkStation – по протоколу TCP/IP обращаются к серверному узлу BankServer.

## **3.2 Диаграмма состояний**

Главное предназначение этой диаграммы — описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий [10].

Диаграмма состояний системы изображена на рисунке 3.2



Рисунок 3.2 – Диаграмма состояний программы

При включении программы она переходит в режим ожидания. Если был введен логин и пароль, то происходит авторизация пользователя. Из этого состояния есть три выхода: неверный ввод логина/пароля, вход под администратором и входи под клиентом. При входах под администратором и под клиентом, программа переходит в состояние выполнения операций, определенных типом доступа, пока не будет выполнена операция закрытия.

## **3.3 Диаграммы последовательностей**

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. Взаимодействия объектов можно рассматривать во времени, и тогда для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности. Взаимодействующие объекты обмениваются между собой некоторой информацией. При этом информация принимает форму законченных сообщений. Другими словами, хотя сообщение и имеет информационное содержание, оно приобретает дополнительное свойство оказывать направленное влияние на своего получателя [10].

На рисунке 3.3 изображена диаграмма последовательности выдачи кредита клиенту. Первоначально клиент обращается к служащему банка с заявлением. Тот в свою очередь при помощи специально программы заполняет данные о клиенте. Далее эти данные анализируются при помощи базы данных репутации клиентов. Если верификация прошла успешно, происходит оформление данных на выдачу кредита в клиентской базе данных. После работник банка получает разрешение на выдачу кредита.

При выдаче депозита клиент обращается в банк с соответствующим заявлением, после чего работник банка запускает специальную программу, при помощи которой заполняются данные, записанные в заявлении. Далее происходит верификация клиента в базе данных клиентов банка. Если после данной процедуры предоставлении услуги не отказано, то работник может оформить открытие депозита, после чего переводит средства клиента в резервы банка (рисунок 3.4).

Рисунок 3.3 – Диаграмма последовательности выдачи кредита



Рисунок 3.4 – Диаграмма последовательности выдачи депозита



## **3.4 Диаграмма классов**

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру (поля, методы) и типы отношений (наследование, реализация интерфейсов). На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

На рисунке 3.5 представлена диаграмма классов разрабатываемой системы. На ней представлена иерархия классов, реализующая бизнес-логику системы.

Одним из основных классов является класс «Пользователь», который является базовым для классов «Клиент» и «СлужащийБанка». Клиент должен оформить заявления, для чего существует одноименный класс. Работник банка обрабатывает/принимает поступающие заявление и по ним оформляет услуги. Класс «Услуга» является базовым для классов «Кредит» и «Депозит». Данные классы имеют свои методы вычисления кредитных выплат, аннуитетных платежей, процентных ставок, а также методы начисления и снятия, закрытия и открытия. Для выполнения последних методов используется интерфейс «Услуга».



Рисунок 3.5 – Диаграмма классов

# **4 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0**

Основным видом операций в разрабатываемой системе является кредитование. Для детального определения функций, которые происходят в процессе реализации кредитования разрабатывается функциональная модель при помощи CASE-средства ProcessModeler на основе стандарта IDEF0 (Приложение А) [11]. На рисунке 4.1 представлен общий вид процесса реализации услуги кредитования.



Рисунок 4.1 – Уровни функциональной модели бизнес-процесса

На контекстной диаграмме верхнего уровня (Приложение А, рисунок А.1) представлена функциональная модель «Реализация услуги кредитования», а также определены потоки входных и выходных данных, механизмы ограничения и управления данными. Как видно из диаграммы, на входе процесса — обращение клиента на предоставление услуги кредитования. Механизмы исполнения — клиент, работники банка, электронные базы данных, поручители, коллеги клиента, автоматизированные системы расчета. Управление — соответствующие правила и нормы, тарифные планы банка, синтаксис и правила работы с БД, статистики, банковские стандарты. На выходе процесса — оформленный кредит.

После описания контекстной диаграммы проводится функциональная декомпозиция — система разбивается на подсистемы, и каждая подсистема описывается отдельно (диаграммы декомпозиции).

На диаграмме декомпозиции первого уровня (Приложение А, рисунок А.2) выделено 3 основных процесса, а именно: принятие заявления на кредит, после которого на выходе появляется подписанное заявление и начинается процесс верификации. Если верификация прошла успешно, то оформляется кредит на клиента.

Принятие заявления состоит из отдельных логически законченных процессов, которые вынесены в декомпозицию второго уровня (Приложение А, рисунок А.3). На данном уровне при обращении клиента подбирается оптимальный тарифный план, на базе которого служащий ознакамливает клиента с условиями кредитования, и, если клиент согласен, оформляется заявление.

Однако на донном уровне также можно выделить пару логических и более детальных цепочек. Одна из них – это подбор оптимального тарифа (Приложение А, рисунок А.4). Работник выясняет предпочтения и возможности клиента; полученные данные систематизируются и анализируются в соответствии с существующими тарифными планами и статистикой кредитования, после чего выявляется оптимальный набор тарифов, составленный по предпочтениям и возможностям клиента. Служащий предъявляет полученный набор клиенту, чтобы тот мог свободно подобрать себе свой кредит.

Оформление заявления включает в себя различные тонкости и пункты, поэтому на функциональной модели данный блок декомпозирован до третьего уровня (Приложение А, рисунок А.5). входным параметром в данном процессе является подтверждение клиента на оформление услуги (то есть при получении подтверждения процесс предоставления кредита остановить будет довольно проблемно для клиента, так как банк – это не отдельный микромир, который существует сам по себе, он связан со всеми сферами жизнедеятельности человека). При получении оного, служащий предоставляет клиенту все необходимые документы для заполнения, после чего заносит полученные данные в электронную базу о клиентах. Далее оформляются готовые экземпляры документов, которые предоставляются клиенту как на подпись, так и для хранения своей копии документа.

Верификация клиентов состоит из нескольких этапов (Приложение А, рисунок А.6). Данные заявлений проверяются по базе данных кредитных историй, проверяются компетентность поручителей и рабочее место клиента, после чего полученные данные анализируются, и, если не выявлено никаких противопоказаний к оформлению кредита, данные подтверждаются и запускается процесс оформления кредита.

В свою очередь оформление кредита является важнейшим этапом при реализации услуги кредитования. Он разбит на несколько пунктов (Приложение А, рисунок А.7):

* в соответствии с условиями тарифного плана и по принципам расчета аннуитетных и дифференцированных платежей вычисляются конкретные процентные ставки и платежи по кредитам для каждого клиента;
* данные по кредиту регистрируются в базе данных по открытым кредитам, со следующего дня по кредиту уже будут начисляться проценты, а также открывается счет;
* на открытый счет клиента переводятся вся сумма по кредиту;
* клиент ознакамливается с правилами и условиями ежемесячных выплат и получает оформленные на него средства.

После завершения всех вышеперечисленных действий процесс по реализации услуги кредитования считается завершенным, так как кредит оформлен.

# **5 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЕ ОПИСАНИЕ**

## **5.1 Краткие теоретические сведения**

Разработанная функциональная модель системы отвечает на вопросы «Что должна делать система?» и «За счет каких действий может быть достигнут требуемый результат?». Эта модель также позволяет концептуально определить наборы данных, используемых в системе.

В то же время она не отвечает на вопрос «Каким образом организованы данные в системе?». Для ответа на него необходима информационная модель.

Традиционно процедуру проектирования базы данных (далее БД) разбивают на три этапа, каждый из которых завершается созданием соответствующей информационной модели: концептуальное, логическое и физическое проектирование.

В настоящее время для проектирования БД активно используются CASE-средства, в основном ориентированные на использование ERD (Entity – Relationship Diagrams, диаграммы «сущность–связь»). С их помощью определяются важные для предметной области объекты (сущности), отношения друг с другом (связи) и их свойства (атрибуты).

Сущность (таблица, в реляционной базе данных (далее РБД) – отношение) – реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению. Если выражаться точнее, то это не объект, а набор объектов (класс) с одинаковыми свойствами. Примеры сущностей: работник, деталь, ведомость, результаты сдачи экзамена и т. д.

Экземпляр сущности (запись, строка, в РБД – кортеж) – уникально идентифицируемый объект.

Связь – некоторая ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. Примерами связей могут являться родственные отношения «отец–сын», производственные – «начальник-подчиненный» или произвольные – «иметь в собственности», «обладать свойством».

Атрибут (столбец, поле) – свойство сущности или связи [12].

## **5.2 Информационная модель розничного бизнеса банка**

Для составления информационной модели розничного бизнеса банка необходимо определить основные сущности:

1 Пользователь. Используется при авторизации приложения, поэтому имеет такие атрибуты, как «пароль», «логин» и «код доступа». Для однозначного определения экземпляра сущности введено поле идентификации (idUser).

2 Администратор и Клиент. Данные сущности связаны с Пользователе отношением категоризации, то есть состоят в иерархии.

3 Кошелек. Как следует из названия, сущность предназначена для хранения средств пользователей. Имеет поле суммы и связана по ключу идентифицирующим отношением с Пользователем.

4 ЗаявкаПоКредиту. Пользователь имеет возможность создавать заявки по кредиту, следовательно, связан с ней идентифицирующим отношением. Так же сущность имеет первичный атрибут – идентификатор заявления, а для своего описания имеет следующие поля: название, цель кредита, фамилия, имя, отчество, место жительства, место работы, сумма кредита и ануитет.

5 ОткрытаяЗаявкаПоКредиту и ЗакрытаяЗаявкаПоКредиту. Данные сущности связаны с ЗаявкойПоКредиту отношением категоризации, и, следовательно, однозначно определены идентификаторами пользователя и заявления.

6 ТарифПоКредиту. Данная сущность несет информативный характер в системе. Когда Пользователь оформляет заявку на кредит, использует в качестве информации именно данную сущность. Однозначно определяется своим кодом и обладает следующими атрибутами: название, цель кредита, минимальная сумма, максимальная сумма, срок (в месяцах), процентная ставка, количество поручителей.

Кроме вышеуказанных сущностей определены «ЗаявкаПоДепозиту», «ОткрытаяЗаявкаПоДепозиту», «ЗакрытаяЗаявкаПоДепозиту» и «ТарифПоДепозиту», которые аналогичны им с различием только оказания услуги, а именно услуги предоставления депозитов.

На рисунке 5.1 представлена составленная информационная модель в соответствии с описанными выше сущностями и их связями.



Рисунок 5.1 – Информационная модель системы розничного бизнеса банка

## **5.3 Нормализация информационной модели**

Для нормализации информационной следует удалить связи N:M, а именно, если в концептуальной модели присутствуют связи N:M, то их нужно устранить путем определения промежуточной сущности. Связь N:M заменяется двумя связями типа 1:M, устанавливаемыми со вновь созданной сущностью. В разработанной системе такие связи отсутствуют, следовательно, этот пункт считается выполненным.

Так же необходимо удалить связи с атрибутами, сложные связи (со степенью участия более 2) и рекурсивные связи (со степенью участия 1).

Не должны присутствовать многозначные атрибуты (атрибуты, имеющие несколько значений). Устраняется путем введения новой сущности и связи 1:N.

Следует избавиться от избыточных связей, а она является таковой, если одна и та же информация может быть получена не только через нее, но и с помощью другой связи.

При выполнении всех вышеперечисленных действий информационная модель системы розничного бизнеса банка приведена к 3-ей нормальной форме (рисунок 5.1).

# **6 ОПИСАНИЕ АЛГОРИТМОВ РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ ПРОЕКТИРУЕМОЙ СИСТЕМЫ**

## **6.1 Алгоритм оформления поступивших заявлений**

Одним из основных алгоритмов бизнес-логики, которые реализует серверную часть приложения является алгоритм оформления поступивших заявлений (рисунок 6.1).

Сервер принимает запрос на оформление заявления услуги кредита или депозита, которое на клиентской части приложения выбрал администратор. Далее должна быть проверка на существование заявления на случай непредвиденной потери частей данных.

При неудачном исходе проверки данных выводится уведомление об отрицательном поиске, и отправляется клиентской части уведомление об ошибке.

При положительном результате найденное заявление добавляется в таблицу открытых текущих услуг. Далее следует изменить состояние заявления на «оформлено» в соответствующей таблице базы данных.

Для кредитов следует вычислить аннуитетные платежи и пересчитать итоговую сумму, которую должен выплатить клиент. Для оформления депозитов следует вычислить начисляемые проценты и пересчитать итоговую сумму депозита. Полученные значения платежей и вкладов следует сохранить в базе данных.

После завершения всех вышеописанных событий сервер сообщает об успешном выполнении и передает данное сообщение клиенту для дальнейшей работы приложения. Далее приведен листинг кода, который реализует данный алгоритм:

**int** idRequest = (Integer)inThread.readObject();  
 System.***out***.println(**"idRequest = "** + idRequest);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);  
 bd.changeStatus(idRequest);  
  
 **int** sumCredit = bd.getSumCodeCredit(idRequest)[0];  
 **int** codeCredit = bd.getSumCodeCredit(idRequest)[1];  
 **int** interestRate = bd.getPersentMonth(codeCredit)[0];  
 **int** period = bd.getPersentMonth(codeCredit)[1];  
  
 **int** annuity = Request.*getAnnuitySum*(sumCredit, interestRate, period);  
 **int** newSum = annuity \* period;  
 System.***out***.println(**"newSum = "** + newSum);  
  
 bd.setAnnuity(annuity, newSum, idRequest);  
 bd.addCurrentCredit(idUser, codeCredit, idRequest);



Рисунок 6.1 – Алгоритм оформления поступивших заявлений

## **6.2 Алгоритм закрытия кредитов**

Бизнес-логика с алгоритмом по оформлению заявлений на услуги банка невозможна без алгоритмов закрытия этих услуг.

На рисунке 6.2 изображен алгоритм оформления закрытия кредита, который начинается с принятия запроса на выполнение. Далее на серверную часть поступает идентификатор заявления, которое требуется закрыть.

В базе данный происходит поиск нужного заявления по полученному идентификатору, и, если результат отрицательный, выводится уведомление, что заявление не найдено и сообщение об ошибке отправляется клиенту.

Если в базе данных существует требуемое заявление, определяется досрочное ли закрытие, и если да, то оставшаяся сумма по кредиту снимается со счета клиента.

После из базы данных текущих кредитов удаляется данное заявление и сохраняется в БД кредитных историй. В базе данных заявлений состояние оформления изменяется на «закрыто».

После указанных процедур сервер выводит сообщение об успешном завершении и передает данное сообщение клиенту. Ниже приведен листинг кода, реализующий данный алгоритм:

**int** idRequest = (Integer)inThread.readObject();  
 System.***out***.println(**"idRequest = "** + idRequest);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);  
 **int** sum = bd.getSumForAllPeriodCredit(idUser);  
 Request request = bd.getClosedCredit(idUser);  
 bd.closedCredit(request);  
 bd.cutMoney(sum, idUser);

## **6.3 Алгоритм закрытия депозитов**

Помимо алгоритма закрытия кредитов бизнес-логика серверной части должна предусмотреть алгоритм по закрытию депозитов, который аналогичен предыдущему.

Сервер получает запрос на закрытия депозита, после чего получает идентификатор депозита, по которому будет производится поиск нужного заявления в базе данных. Если поиск не дал результатов, то выводится сообщение об ошибке и отправляется на клиентскую часть приложения.

После нахождения нужного заявления на счет клиенту переводится вся сумма вклада. А далее из базы данный открытых вкладов извлекается данное заявление и сохраняется в БД истории депозитов.

При завершении всех действий, указанных выше, сервер выводит сообщение об успешности закрытия депозита и передает его на клиентскую сторону.



Рисунок 6.2 – Алгоритм оформления закрытия кредита

## **6.3 Обобщенный алгоритм работы серверной части системы**

Перечисленные выше алгоритмы выполнения определенных операций являются всего лишь частью большой системы, они включены в общий алгоритм работы серверной части системы, который представлен на рисунке 6.3.

При включении сервера он создает сокет, посредством которого прослушивает все приходящие запросы. Открывается цикл по подключению клиентов, который заканчивается при выключении сервера. Если подключается клиент, то для него создается новый поток, для того, чтобы одновременно могли подключаться множество клиентов, и они могли нормально, без перебоев работать. В отдельном потоке создается цикл, в котором сервер прослушивает запросы от определенно клиента.

Сервер принимает запрос на выполнение какой-либо операции, и обрабатывает его в соответствии с видом операции, на которую пришел запрос. Если клиент ждет ответа, например, каких-либо данных, запрос на получение которых пришел, то сервер передает эти данные. После этих операций сервер переходит в состояние прослушивания запросов от клиента на новые операции. Цикл заканчивается, если клиент отключился.

После завершения всей работы и перед выключением сервера происходит закрытие сокета.



Рисунок 6.3 – Общий алгоритм работы серверной части системы

# **7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Для функционирования программы необходима БД MySQL. Для ее генерации можно воспользоваться sql-скриптом – db\_bank.sql. Далее требуется запустить сервер приложения – server.jar. Программа клиента, которая предназначена для пользователей запускается файлом client.jar.

При запуске программы, а именно клиентской части приложения, открывается окно авторизации (рисунок 7.1), где пользователю предлагается зайти в приложение под своим логином и паролем, если он уже зарегистрирован, и зарегистрироваться, если ранее клиент не пользовался данным приложением.

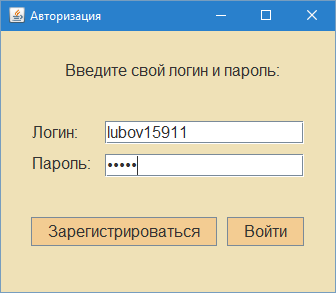


Рисунок 7.1 – Окно авторизации

После авторизации, если пользователь является клиентом, открывается окно пользователя (рисунок 7.2), в котором выводится основная информация о состоянии, а именно, открыл ли какой-нибудь депозит или кредит и основная информация по ним. Если пользователь отправлял заявки на оформление какой-либо услуги, то в окне так же указывается, что заявление находится на рассмотрении.

В окне клиента с левой стороны расположено меню. В пункте меню «Кошелек» клиент имеет возможность просмотреть сумму (в рублях), которая хранится на его счету. Так же клиент может пополнить счет, введя необходимую сумму в поле ввода, и выполнить запрос нажатием на клавишу «Добавить» (рисунок 7.3).

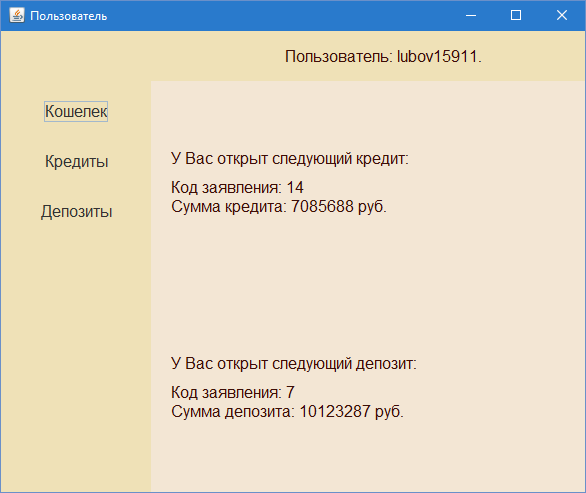


Рисунок 7.2 – Главное окно клиента

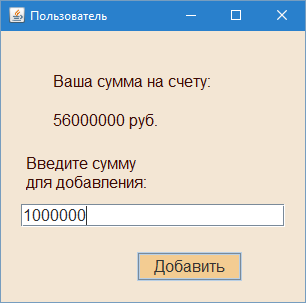


Рисунок 7.3 – Окно кошелька и пополнения счета

При выборе пункта меню «Кредиты» открывается основное меню по услугам кредитования (рисунок 7.4). Клиент не может воспользоваться услугой «Погашения кредита», если он не оформлял заявок или его заявка находится на рассмотрении (рисунок 7.5).

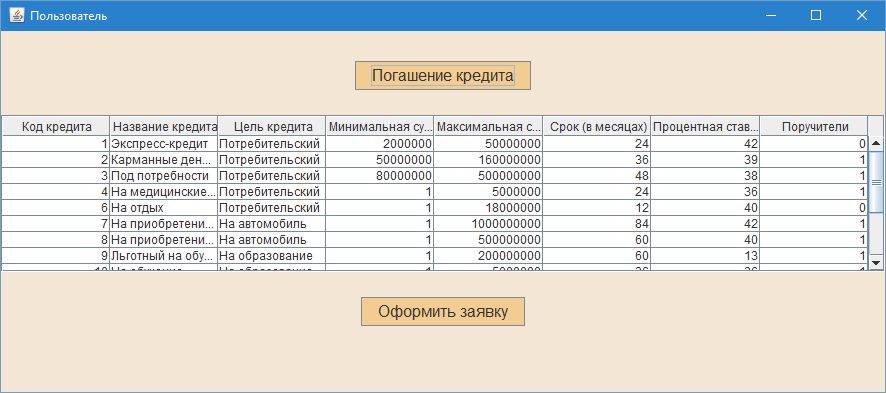


Рисунок 7.4 – Окно услуг кредитования

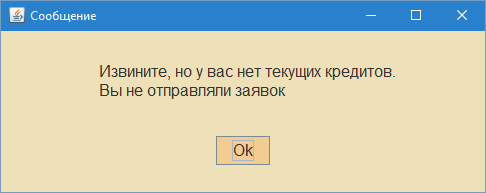


Рисунок 7.5 – Сообщение о невозможности погашения кредита

Если же заявление существует, и оно оформлено, то открывается окно погашения кредита (рисунок 7.6). На нем клиенту выводится информация об аннуитетном платеже, который нужно выплачивать каждый месяц, информация об общей сумме, которую клиент обязан выплатить по истечении срока кредита, и, для удобства, сразу показана сумма на счету у клиента.

Решив выплатить сумму за месяц, клиент должен нажать на кнопку «За месяц», после чего с его счета снимается сумма аннуитетного платежа. При оплате сразу всей суммы за весь срок нужно, чтобы на счету была необходимая сумма или больше, и после выбора соответствующей операции формируется заявка на досрочное погашение кредита, которая потом отправляется в банк на рассмотрение (рисунок 7.7).

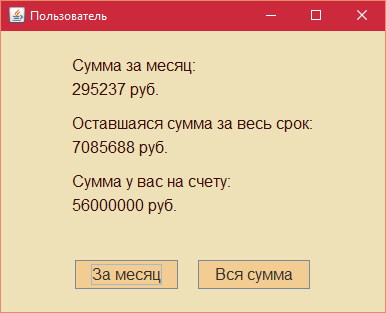


Рисунок 7.6 – Окно погашения кредита

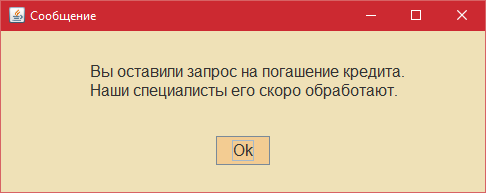


Рисунок 7.7 – Сообщение об отправке запроса на досрочное погашение оставшейся суммы по кредиту

В главной части окна расположена таблица со всеми кредитными планами, которые существуют в банке. Проанализировав данную таблицу и выбрав оптимальный для него план, клиент может оформить заявление на кредит нажатием по кнопке «Оформить заявку», после чего открывается окно оформления заявления (рисунок 7.8).

После отправки заявления клиент должен будет ожидать, пока работники банка не оформят его заявление.

Пункт «Депозиты». Как и при кредитовании по нажатию на данный пункт меню открывается главное окно по предоставлению депозитов (рисунок 7.9).

Если пользователь не оформлял никаких вкладов, или его заявление еще не было рассмотрено, то при нажатии на кнопку «Закрытие вклада», клиенту открывается уведомление о невозможности совершения операции (рисунок 7.10).

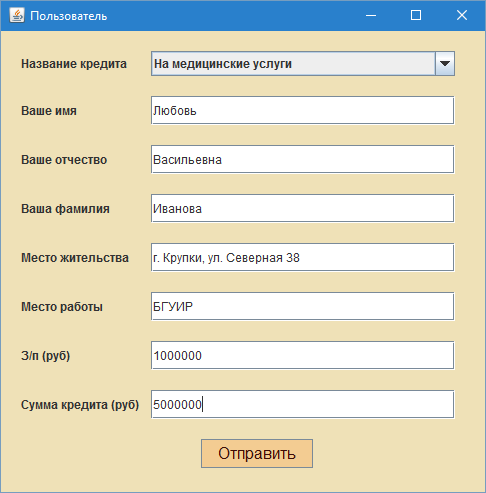


Рисунок 7.8 – Оформление заявления на услугу кредитования

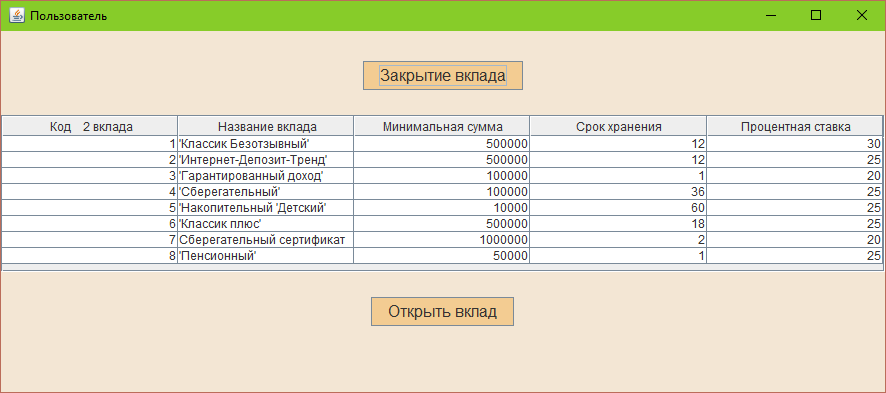


Рисунок 7.9 – Окно по предоставлению услуг по депозитам

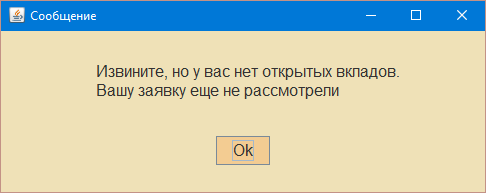


Рисунок 7.10 – Предупреждение невозможности выполнения операции

Операция закрытия депозита произойдет, если пользователь уже открывал вклад. При этом выпоре данной операции клиентом формируется заявка на закрытие депозита, и открывается окно уведомления, в котором сообщается о том, что заявление сформировано, но нужно подтверждение со стороны работников банка (рисунок 7.11).

На главном окне депозитов в центре можно увидеть перечень всех предоставляемых клиентам депозитов. Это информация для ознакомления и возможного дальнейшего руководства при выборе, какой именно вклад выгодно открыть клиенту.

Клиент может оформить заявление, если определился с видом вклада, нажав на кнопку «Открыть вклад». Откроется окно заявления на открытия вклада (рисунок 7.12).

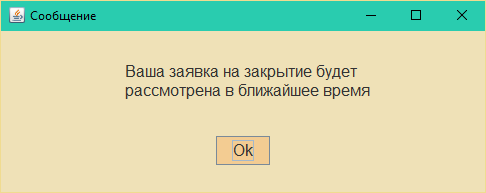


Рисунок 7.11 – Уведомление о запросе на закрытие депозита

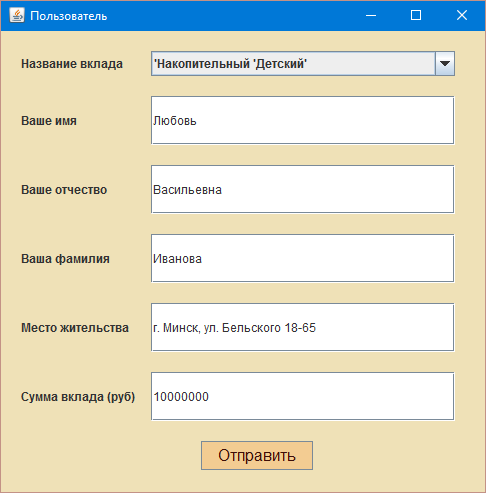


Рисунок 7.12 – Оформление заявления на открытие вклада

После отправки заявления, и последующего его оформления со стороны работников банка, клиент имеет счет с заявленной в заявлении суммой.

Данной программой можно также пользоваться и работникам банка. Если ввести соответствующий логин и пароль, то для служащего откроется главное окно администратора (рисунок 7.13).

Структура данного окна представляет собой меню с операциями, которые может выполнить работник. В меню выделены две логические части: для работы с клиентами и для работы с тарифными планами и их анализом.

Часть меню по работе с клиентами включает в себя следующие пункты меню:

1. Заявления на кредиты;
2. Закрытие кредитов;
3. Заявления на вклады;
4. Закрытие вкладов.

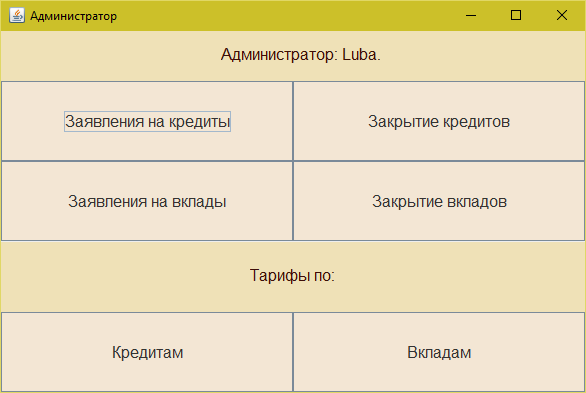


Рисунок 7.13 – Главное окно администратора

При оформлении заявлений на кредиты следует выбрать соответствующий пункт меню, после чего открывается окно по работе с заявлениями на кредиты (рисунок 7.14).

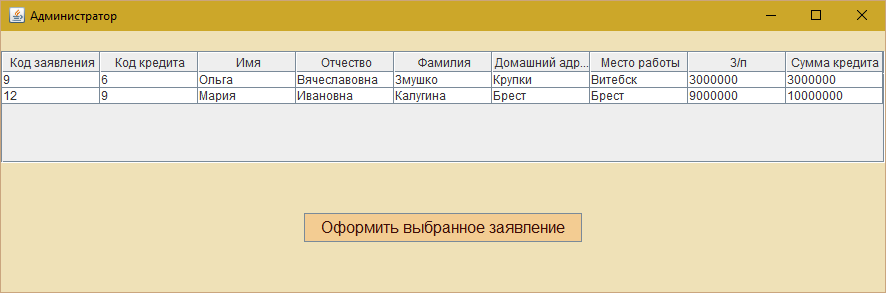


Рисунок 7.14 – Окно оформлений заявлений на кредиты

Чтобы оформить какое-либо заявление, следует выбрать его в таблице, которая расположена в главной части окна, и нажать кнопку «Оформить выбранное заявление». После выполнения данной операции в таблице заявлений, требующих оформления, больше не будет выводится выбранное заявление, так как оно перешло в статус оформленных. Клиент с этого момента может при помощи данного приложения погашать кредит.

При желании закрыть кредит следует выбрать соответствующий пункт меню в главном окне администратора. На рисунке 7.15 изображено окно, которое открывается, если выбрать этот пункт.

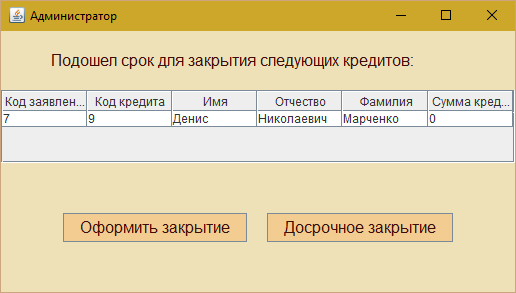


Рисунок 7.15 – Окно закрытия кредитов

В таблице выводятся те кредиты, которые клиенты уже выплатили, и которые следует закрыть, чтобы клиент мог в следующий раз при желании еще раз оформить какой-либо кредит в банке. Для оформления таких кредитов следует выбрать заявление в таблице и нажать на кнопку «Оформить закрытие».

Так как клиенты имеют возможность досрочно погасить оставшуюся сумму кредита, работники банка должны отслеживать такие запросы. Чтобы посмотреть, какие клиенты обратились с досрочным погашением, следует нажать на кнопку «Досрочное закрытие», при этом открывается аналогичное окно (рисунок 7.16), но в таблице выводится список заявлений клиентов о досрочном погашении.

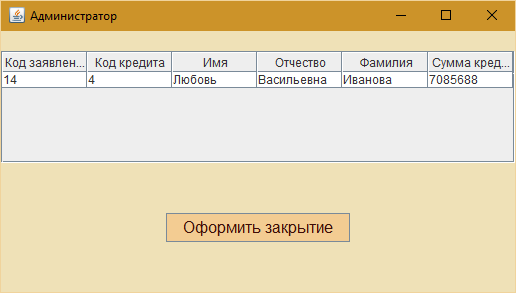


Рисунок 7.16 – Окно заявлений о досрочном погашении кредита

Как и описывалось выше, для закрытия кредита в таблице следует выбрать заявление и нажать на кнопку «Оформить закрытие».

Оставшиеся два пункта в первой части меню работают с депозитами клиентов. Для открытия депозита нужно выбрать пункт «Заявления на вклады». Откроется окно со списком заявлений от клиентов, которые желают открыть депозит (рисунок 7.17).

Чтобы оформить заявление на открытие вклада необходимо выбрать это заявление в таблице и начать на кнопку «Оформить выбранное заявление».

В случае, если клиент желает закрыть вклад и забрать деньги, он отправляет соответствующее заявление. Для просмотра этих сообщений в главном меню администратора нужно выбрать пункт «Закрытие вкладов» (рисунок 7.18).

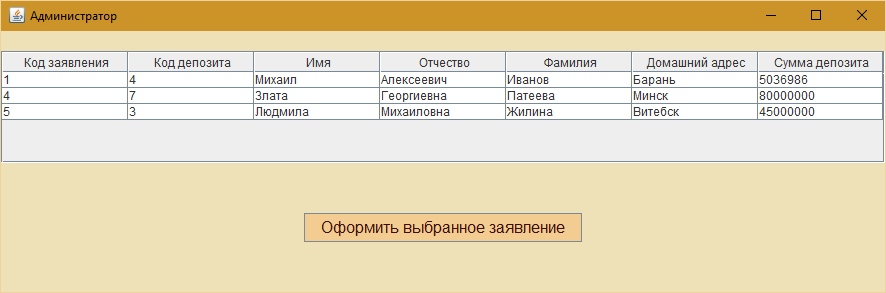


Рисунок 7.17 – Окно оформления депозитов

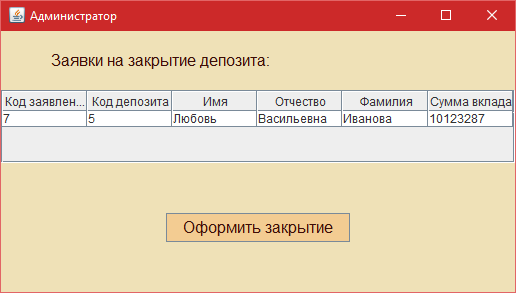


Рисунок 7.18 – Окно заявлений на закрытие депозита

При выборе в списке заявления и нажатии на соответствующей кнопке происходит закрытие депозита. После выполнения это операции клиент может увидеть у себя на счету сумму, указанную в оформленном заявлении.

Для работы с тарифными планами в главном меню администратора есть пункт «Кредиты» и «Вклады». При работе с кредитами открывается соответствующее окно (рисунок 7.19).

В верхней части окна расположены три кнопки. Кнопка «Редактирование» позволяет вносить изменения по тарифным планам прямо в таблице (рисунок 7.20), а для сохранения внесенных изменений следует выбрать кнопку «Сохранить».



Рисунок 7.19 – Окно для работы с тарифами кредитов

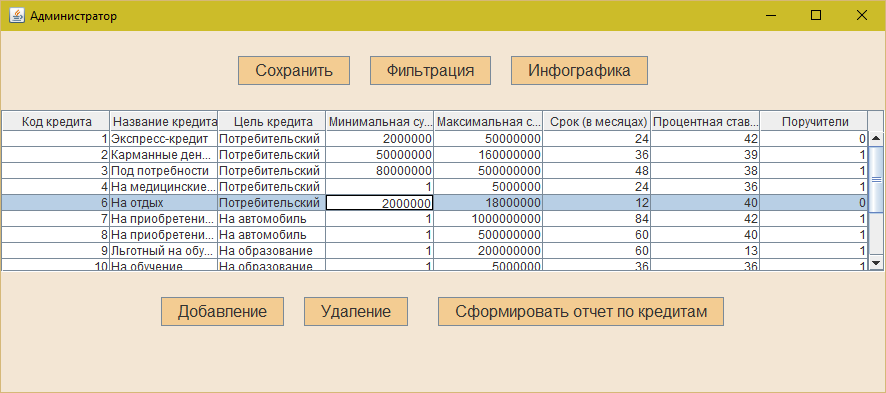


Рисунок 7.20 – Редактирование данных по тарифам

Чтобы отфильтровать и оставить в списке какую-то определенную группу тарифов можно выбрать кнопку «Фильтрация» (рисунок 7.21).

Приложение предоставляет возможность анализа данных в графическом представлении. Для этого предусмотрена кнопка «Инфографика». При нажатии на нее в окне будет получен график, на котором показана популярность тарифным планов, а именно, по каким тарифам сколько заявлений было оформлено (рисунок 7.22).

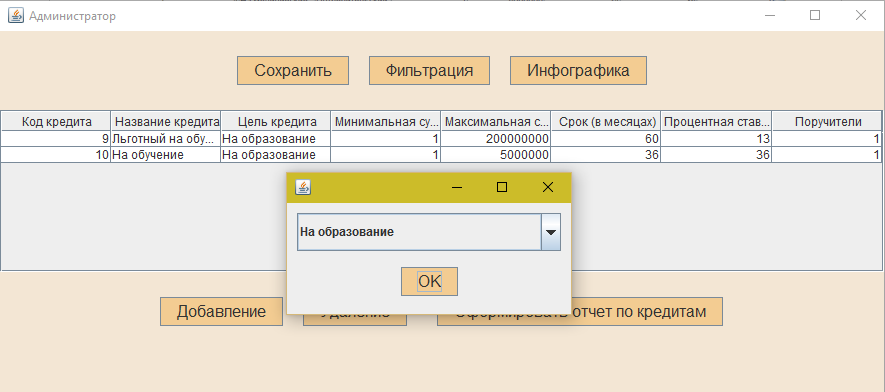


Рисунок 7.21 – Фильтрация тарифов по цели кредита (на образование)

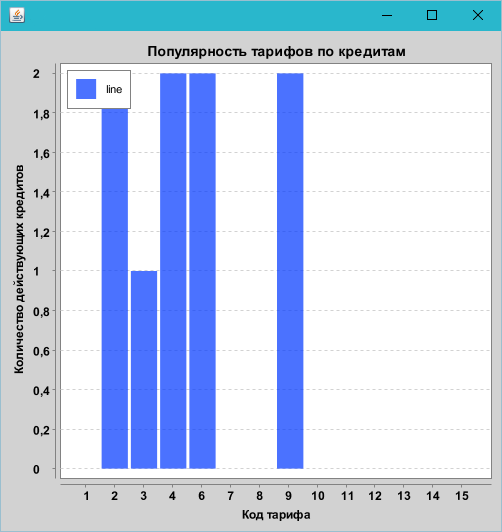


Рисунок 7.22 – График популярности тарифных планов

В нижней части окна с тарифными планами также есть три кнопки. Первая из них отвечает за удаление данных из таблицы, вторая – за добавление нового тарифного плана.

Для удаления данных следует в таблице выбрать тарифный план и нажать на кнопку удаления.

При нажатии кнопки добавления открывается окно с данными для заполнения, чтобы составить новый тарифный план (рисунок 7.23).

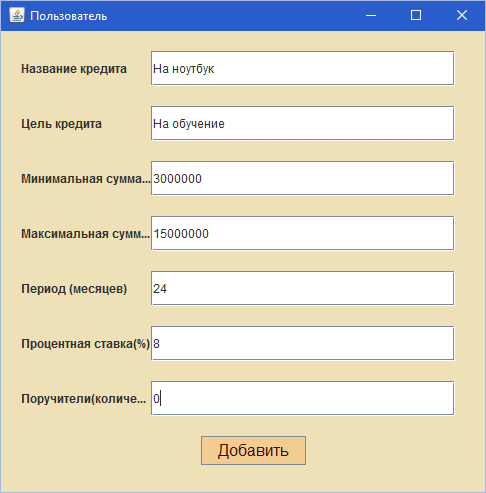


Рисунок 7.23 – Окно составления нового тарифного плана

Третья кнопка на нижней панели – «Сформировать отчет по кредитам» – отвечает за составление текстового отчета по всем заявлениям, а именно: заявления по текущим кредитам, заявления, которые ожидают оформление и заявления кредитов, которые уже погашены. На рисунке 7.24 изображена часть отчета.

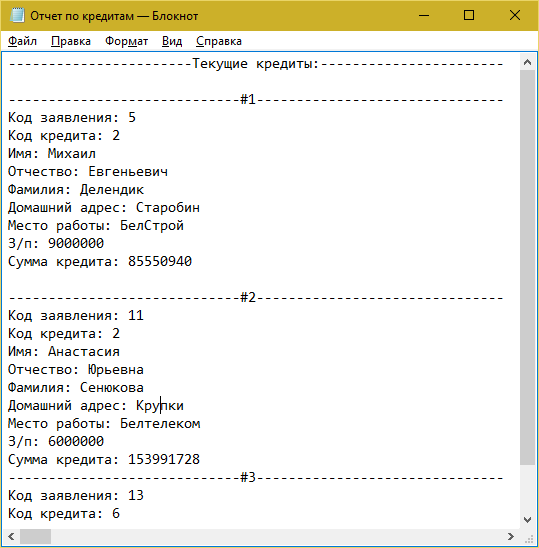


Рисунок 7.24 – Отчет по заявлениям на кредиты

Для работы с тарифами по депозитам существует пункт меню «Вклады» в главном окне администратора. Вид окон при работе с депозитами и назначения кнопок аналогичны окнам и кнопкам при работе с кредитами.

# **8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ**

Несмотря на то, что роль тестирования на первый взгляд может показаться не столь уж значительной, процесс тестирования программного обеспечения представляет собой столь же неотъемлемую часть разработки, как и проектирование. Тестирование является составляющей частью процесса отладки ПО, после выявления ошибок дефекты в программном коде должны быть устранены разработчиками.

Учитывая этот важный процесс разработки, разработанная система тестируется на самые разные варианты ошибок. Далее приводятся результаты обработки таких ошибок в разработанной системе.

При авторизации пользователь может случайно не ввести поля, которые обязательны, в таком случае появляется уведомление, что заполнить нужно все, при этом никакие данные на сервер не отправляются (рисунок 8.1).

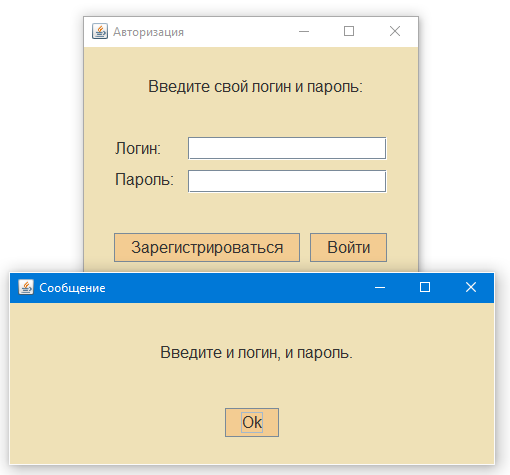


Рисунок 8.1 – Сообщение об ошибке если заполнены не все поля для авторизации

Если пользователь вводит неверный логин либо пароль, и пытается войти в программу, это будет невозможно (рисунок 8.2).

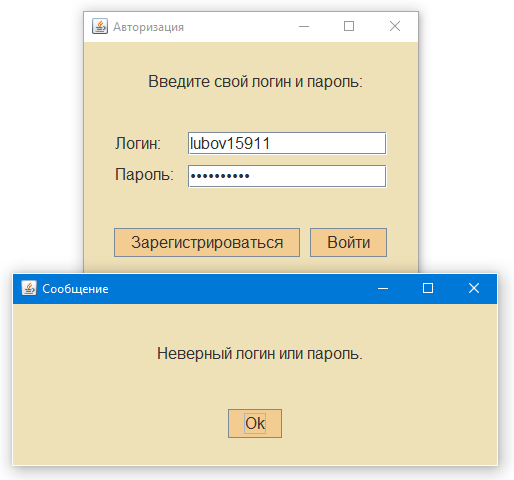


Рисунок 8.2 – Сообщение о неправильности введенных данных при авторизации

Зарегистрировать пользователя с логином, который уже существует в базе данных невозможно (рисунок 8.3).

В программе существует большое количество различных полей для ввода данных. Для предотвращения ввода символов в поля, предназначенные для цифровых значение, стоит ограничение на ввод ненужных символов. Таким образом система избегает ошибок, связанных с разными типами данных.

При обработке текстовых следует также проверять логичность вводимых клиентом данных. Например, при оформлении заявления на кредит сумма оформляемого кредита не может быть меньше, чем заявленная минимальная сумма тарифного плана, который выбрал клиент и больше, чем максимальная сумма в тарифном плане (рисунок 8.4).

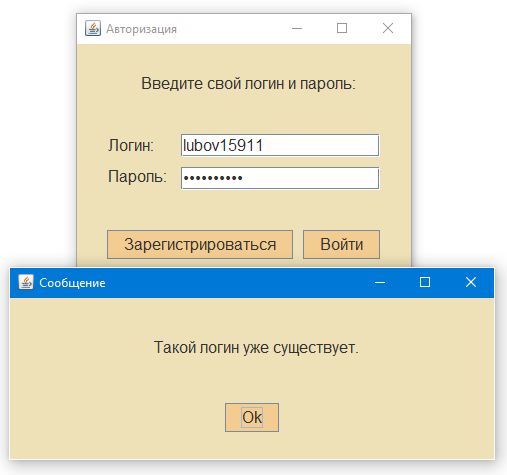


Рисунок 8.3 – Сообщение об ошибке при регистрации пользователя с одинаковыми логинами

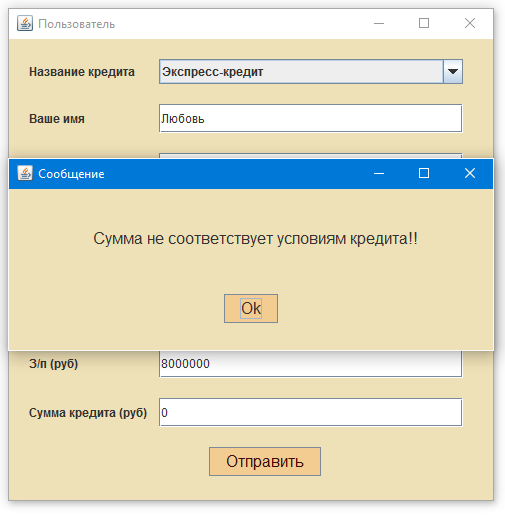


Рисунок 8.4 – Ошибка при оформлении суммы кредита

Логично, что предоставить кредит можно только пользователю с хорошей и стабильной заработной платой (рисунок 8.5)

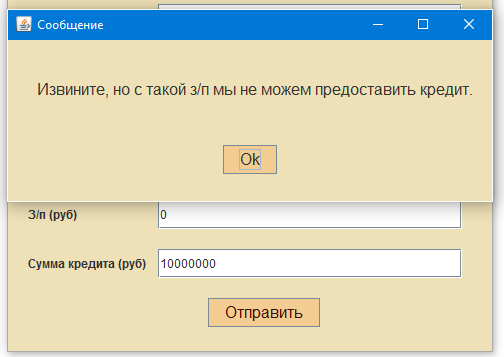


Рисунок 8.5 – Сообщение об ошибке при неподходящей заработной плате для оформления

К логическим проверкам так же относятся возможность или невозможность оформления кредита или депозита на пользователя. Он не может отправить огромное количество заявок на кредит (рисунок 8.6). Также клиент не может иметь несколько кредитов, поэтому при попытке оформить еще один кредит будет показано сообщение (рисунок 8.7)

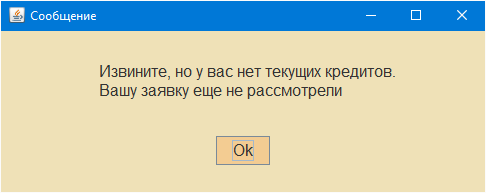


Рисунок 8.6 – Сообщение о невозможности повторного отправления заявки

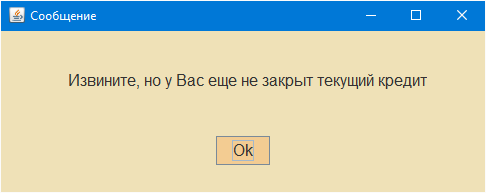


Рисунок 8.7 – Ошибка при оформлении еще одного кредита

С точки зрения открытия депозитов данные проверки идентичные.

Клиент при погашении кредита не может оплатить нужную сумму, если в кошельке лежит меньшее количество средств (рисунок 8.8).

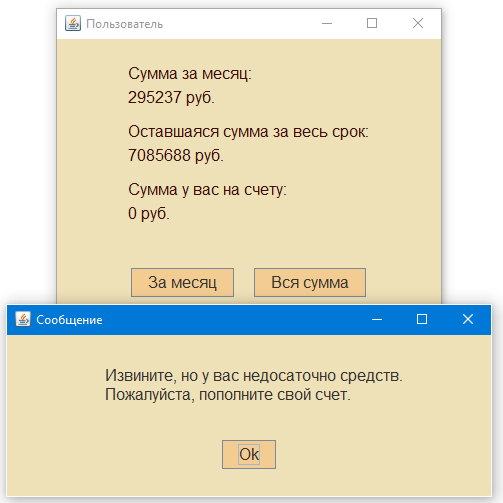


Рисунок 8.8 – Ошибка погашения кредита при меньшей сумме на счету

При тестировании проблемных мест программы исключительные ошибки не выявлены, из чего можно заключить, что система работает стабильно.

**ЗАКЛЮЧЕНИЕ**

Автоматизированная банковская система для розничного бизнеса концентрирует автоматизацию всего спектра банковских операций, которые могут быть востребованы при продаже или оказании услуг клиентам физическим лицам – это вклады, кредиты, операции с пластиковыми карточками, переводы, платежи, обмен валюты. При этом выполняется полный цикл автоматизации по всем направлениям банковской деятельности в секторе обслуживания клиентов. Также система позволяет осуществлять анализ текущей деятельности и формировать прогнозные показатели, обеспечивая руководство банка необходимой управленческой информацией.

В ходе выполнения работы были выполнены поставленные задачи посредством детального рассмотрения и изучения розничного бизнеса банка, основных принципов кредитования и предоставления депозитов клиентам.

На основе полученных знаний разрабатывалась архитектура проекта с помощью средств UML, стандартов IDEF0 и IEDF1X. Изучены основные паттерны проектирования на языке Java, и из них в разработке применены наиболее подходящие в соответствии с предметной областью.

Разработанное приложение является клиент-серверным приложением и для реализации данного вида приложения рассмотрены основные средства сетевого программирования.

Приложение обладает интерфейсом, разработанным средствами библиотеки Swing, что является довольно удобным решением, принятым с заботой о будущих пользователях программного продукта.

В соответствии с поставленными задачами, система обладает различными возможностями, которые максимально автоматизируют процессы оказания услуг в сфере розничного бизнеса банка. Работники банка только контролируют, запускают и останавливают процессы по реализации услуг.

Для клиента возможность дистанционного взаимодействия с банком в наше время является наиболее привлекательным решением при разработке.

Подведя итоги, можно заключить, что разработанный программный продукт соответствует всем заявленным критериям и является точным решением по автоматизации розничного бизнеса банка.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

[1] Бюллетень банковской статистики № 9 (183) // Национальный банк Республики Беларусь [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www. nbrb.by/publications/bulletin/Bulletin2014\_09.pdf.

[2] Стефанович, Л. И. Карточный рынок Беларуси: карточки, инфраструктура, операции / Л. И. Стефанович // Банкаўскi веснiк [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.nbrb.by/bv/articles/10027.pdf.

[3] Итуэлл Дж., Милгейт М. Экономическая теория. Учебное пособие. – М: «ИНФРА», 2004 – 944с.

[4] Хорстманн, К. С. JAVA2. Том 1. Основы. Пер. с англ. Справочное пособие/ Корнелл, Г. – М: ООО «Вильямс», 2003 – 849с.

[5] Хорстманн, К. С. JAVA2. Том 2. Тонкости программирования. Пер. с англ. Справочное пособие/ Корнелл, Г. – М: ООО «Вильямс», 2003 – 1120с.

[6] Шилдт, Г. "Java. Полное руководство. 8-е издание". – М: Вильямс.

[7] Крейг Хант.TCP/IP. Сетевое администрирование. Пер. с англ. Отдельное издание/ Зислис, М. – М: «Символ-Плюс», 2006 – 816с.

[8] Гамма, Э. Приемы объектно-ориентированного программирования. Паттерны проектирования. Справочное пособие/ Р. Хелм, Р.Джонсон. – М.: Питер, 2010. — С. 368.

[9] wikipedia.org [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Шаблон\_проектирования

[10] Крэг Ларман. Применение UML и шаблонов проектирования. 2-е издание. Пер. с англ. – М.: Издательский дом «Вильямс», 2004. — 624 с.

[11] nsu.ru [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.nsu.ru/smk/files/idef.pdf

[12] i-novice.net [Электронный ресурс]. – Электронные данные. – Режим доступа: http://i-novice.net/6-normalnyx-form-bd/

[13] Шилдт, Г. "Библиотека SWING для Java: руководство для начинающих". – М: Вильямс, 2007. – 704 c.

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Функциональная модель**

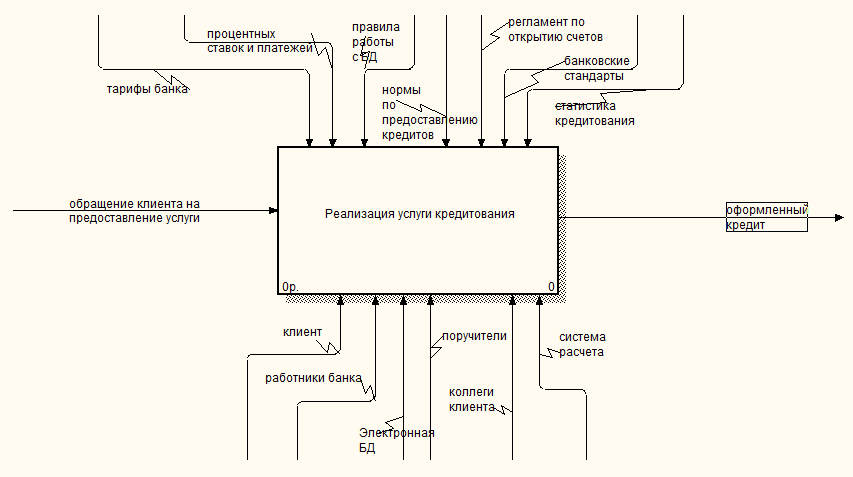
****

Рисунок А.1 – Контекстная диаграмма

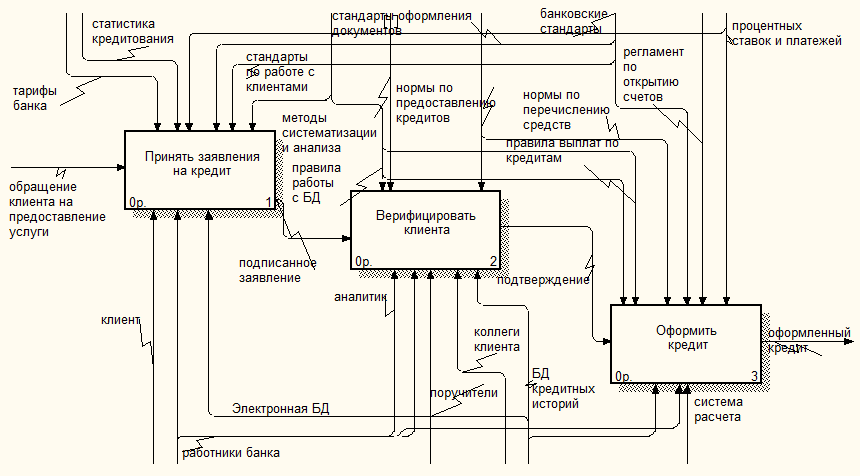


Рисунок А.2 – Декомпозиция первого уровня

Продолжение приложения А

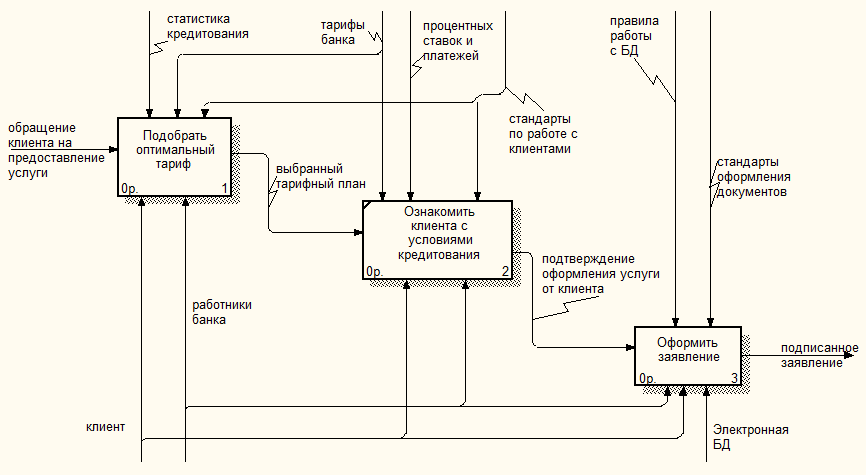


Рисунок А.3 – Декомпозиция второго уровня блока «Принятия заявление на кредит»

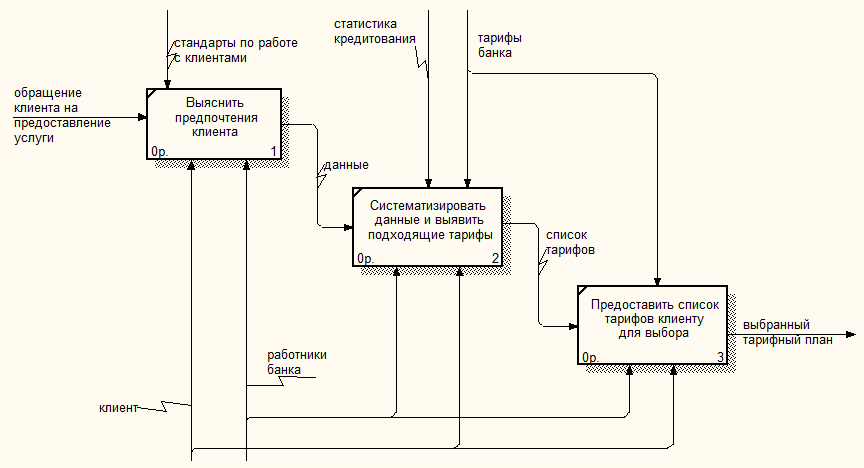


Рисунок А.4 – Декомпозиция третьего уровня блока «Подобрать оптимальный вариант»

Продолжение приложения А

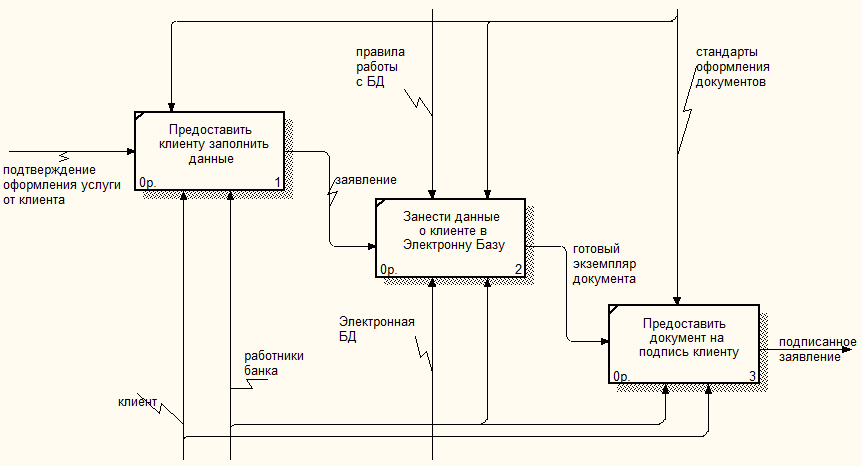


Рисунок А.5 – Декомпозиция третьего уровня блока «Оформить заявление»

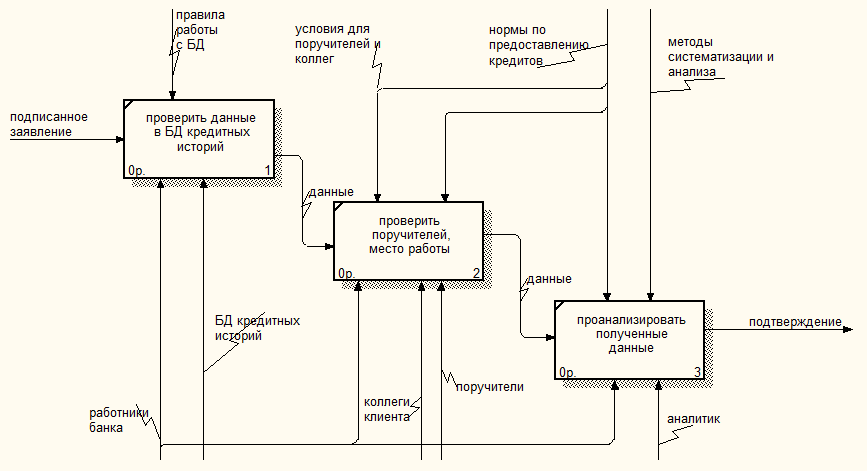


Рисунок А.6 – Декомпозиция второго уровня блока «Верифицировать клиента»

Продолжение приложения А

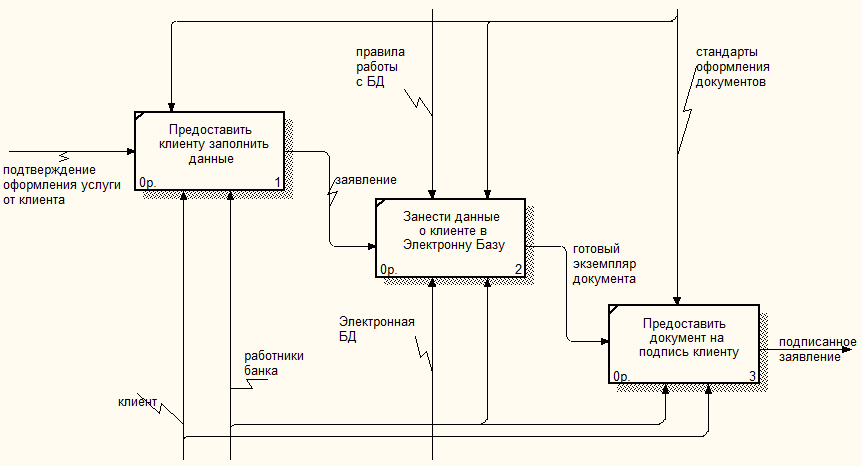


Рисунок А.7 – Декомпозиция второго уровня процесса оформления кредита

# **ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Листинг основных элементов программы**

Класс Client (для прослушивания запросов со стороны клиента):

**public class** Client **implements** Runnable {  
 **private** Socket **clientAccepted**;  
 **private** Thread **thread**;  
  
 **public** Client(Socket clientAccepted) {  
 **this**.**clientAccepted** = clientAccepted;  
 **thread** = **new** Thread(**this**);  
 **thread**.start();  
 }  
  
 @Override  
 **public void** run() {  
 ObjectInputStream inThread = **null**;  
 ObjectOutputStream outThread = **null**;  
 **try** {  
 inThread = **new** ObjectInputStream(**clientAccepted**.getInputStream());  
 outThread = **new** ObjectOutputStream(**clientAccepted**.getOutputStream());  
  
 DB bd = **new** DB();  
 **boolean** flag = **true**;  
 **while** (flag) {  
 System.***out***.println(**"2 "** + flag);  
 **int** access = (Integer)inThread.readObject();  
 System.***out***.println(**"access "** + access);  
 **switch** (access) {  
 **case** 0: *// sign in* {  
 String login = (String)inThread.readObject();  
 String password = (String)inThread.readObject();  
 User user = bd.CheckData(login, password);  
 **if** (user != **null**) {  
 System.***out***.println(**"access sign in "** + user.getAccess());  
 outThread.writeObject(user.getLogin());  
 outThread.writeObject(user.getPassword());  
 outThread.writeObject(user.getAccess());  
 outThread.writeObject(user.getId());  
 }  
 **break**;  
 }  
 **case** 1: *// registr* {  
 String login = (String)inThread.readObject();  
 System.***out***.println(**"login "** + login);  
 String password = (String)inThread.readObject();  
 System.***out***.println(**"password "** + password);  
 **int** success = bd.Registration(login, password);  
 outThread.writeObject(success);  
 **break**;  
 }  
 **case** 2: *// get gross assets* {

Продолжение приложения Б

System.***out***.println(**"get gross assets "**);  
 **int** id = (Integer)inThread.readObject();  
 System.***out***.println(**"id = "** + id);  
 **int** grossAssets = bd.getGrossAssets(id);  
 System.***out***.println(**"grossAssets "** + grossAssets);  
 outThread.writeObject(grossAssets);  
 **break**;  
 }  
 **case** 3: *// add money* {  
 **int** id = (Integer)inThread.readObject();  
 System.***out***.println(**"id = "** + id);  
 **int** sum = (Integer)inThread.readObject();  
 System.***out***.println(**"sum = "** + sum);  
 bd.addMoney(id, sum);  
 **break**;  
 }  
 **case** 4: *// get tariffs* {  
 **int** numberTariffs = bd.getNumberTariffs();  
 System.***out***.println(**"numberTariffs = "** + numberTariffs);  
 outThread.writeObject(numberTariffs);  
 ArrayList<Tariff> listTariff = bd.getTariffs();  
  
 **for** (**int** i = 0; i < numberTariffs; i++) {  
 outThread.writeObject(listTariff.get(i).getCode());  
 outThread.writeObject(listTariff.get(i).getTitle());  
 outThread.writeObject(listTariff.get(i).getLoanPurpose());  
 outThread.writeObject(listTariff.get(i).getMinSum());  
 outThread.writeObject(listTariff.get(i).getMaxSum());  
 outThread.writeObject(listTariff.get(i).getPeriod());  
 outThread.writeObject(listTariff.get(i).getInterestRate());  
 outThread.writeObject(listTariff.get(i).getWarrenters());  
 }  
 **break**;  
 }  
 **case** 5: *// get Count Request Credits* {  
 **int** id = (Integer)inThread.readObject();  
 System.***out***.println(**"id = "** + id);  
  
 **int** countRequestCredits = bd.getCountRequestCredits(id);  
 System.***out***.println(**"countRequestCredits = "** + countRequestCredits);  
  
 outThread.writeObject(countRequestCredits);  
 **break**;  
 }  
 **case** 6: *// get Count Request Deposits* {  
 **int** id = (Integer)inThread.readObject();

Продолжение приложения Б

System.***out***.println(**"id = "** + id);  
  
 **int** countRequestDeposits = bd.getCountRequestDeposits(id);  
 System.***out***.println(**"countRequestDeposits = "** + countRequestDeposits);  
  
 outThread.writeObject(countRequestDeposits);  
 **break**;  
 }  
 **case** 7: *// send Request Credit* {  
 **int** id = (Integer)inThread.readObject();  
 System.***out***.println(**"id = "** + id);  
 **int** code = (Integer)inThread.readObject();  
 System.***out***.println(**"code = "** + code);  
 String name = (String)inThread.readObject();  
 System.***out***.println(**"name = "** + name);  
 String surname = (String)inThread.readObject();  
 System.***out***.println(**"surname = "** + surname);  
 String lastname = (String)inThread.readObject();  
 System.***out***.println(**"lastname = "** + lastname);  
 String houseAddress = (String)inThread.readObject();  
 System.***out***.println(**"houseAddress = "** + houseAddress);  
 String jobAddress = (String)inThread.readObject();  
 System.***out***.println(**"jobAddress = "** + jobAddress);  
 **int** salary = (Integer)inThread.readObject();  
 System.***out***.println(**"salary = "** + salary);  
 **int** sumCredit = (Integer)inThread.readObject();  
 System.***out***.println(**"sumCredit = "** + sumCredit);  
  
 Request request = **new** Request(id, code, name, surname, lastname, houseAddress, jobAddress, salary, sumCredit);  
  
 bd.sendRequestCredit(request);  
 **break**;  
 }  
 **case** 8: *// get Current Credit* {  
 **int** id = (Integer)inThread.readObject();  
 System.***out***.println(**"id = "** + id);  
  
 **int** status = bd.getCurrentCredit(id);  
 System.***out***.println(**"currentCredit status = "** + status);  
  
 outThread.writeObject(status);  
 **break**;  
 }  
 **case** 9: *// get Current Deposit* {  
 **int** id = (Integer)inThread.readObject();  
 System.***out***.println(**"id = "** + id);  
  
 **int** status = bd.getCurrentDeposit(id);  
 System.***out***.println(**"CurrentDeposit status = "** + status);  
  
 outThread.writeObject(status);  
 **break**;  
 }

Продолжение приложения Б

**case** 10: *// get deposits* {  
 **int** numberDeposits = bd.getNumberDeposits();  
 System.***out***.println(**"numberDeposits = "** + numberDeposits);  
 outThread.writeObject(numberDeposits);  
 ArrayList<Deposit> listDeposit = bd.getDeposits();  
  
 **for** (**int** i = 0; i < numberDeposits; i++) {  
 outThread.writeObject(listDeposit.get(i).getCode());  
 outThread.writeObject(listDeposit.get(i).getTitle());  
 outThread.writeObject(listDeposit.get(i).getMinSum());  
 outThread.writeObject(listDeposit.get(i).getPeriod());  
 outThread.writeObject(listDeposit.get(i).getInterestRate());  
 }  
 **break**;  
 }  
 **case** 11: *// send Request Deposit*

{  
 **int** id = (Integer)inThread.readObject();  
 System.***out***.println(**"id = "** + id);  
 **int** code = (Integer)inThread.readObject();  
 System.***out***.println(**"code = "** + code);  
 String name = (String)inThread.readObject();  
 System.***out***.println(**"name = "** + name);  
 String surname = (String)inThread.readObject();  
 System.***out***.println(**"surname = "** + surname);  
 String lastname = (String)inThread.readObject();  
 System.***out***.println(**"lastname = "** + lastname);  
 String houseAddress = (String)inThread.readObject();  
 System.***out***.println(**"houseAddress = "** + houseAddress);  
 **int** sumDeposit = (Integer)inThread.readObject();  
 System.***out***.println(**"sumDeposit = "** + sumDeposit);  
  
 Request request = **new** Request(id, code, name, surname, lastname, houseAddress, sumDeposit);  
  
 bd.sendRequestDeposit(request);

**break**;  
 }  
 **case** 12: *// get Requests Credit* {  
 **int** numberRequests = bd.getNumberRequestsCredit();  
 System.***out***.println(**"numberRequests = "** + numberRequests);  
 outThread.writeObject(numberRequests);  
 ArrayList<Request> listRequests = bd.getRequestsCredit();  
  
 **for** (**int** i = 0; i < numberRequests; i++) {  
 outThread.writeObject(listRequests.get(i).getIdRequest());  
 outThread.writeObject(listRequests.get(i).getId());

Продолжение приложения Б

outThread.writeObject(listRequests.get(i).getCode());  
 outThread.writeObject(listRequests.get(i).getName());  
 outThread.writeObject(listRequests.get(i).getSurname());  
 outThread.writeObject(listRequests.get(i).getLastname());  
 outThread.writeObject(listRequests.get(i).getHouseAddress());  
 outThread.writeObject(listRequests.get(i).getJobAddress());  
 outThread.writeObject(listRequests.get(i).getSalary());  
 outThread.writeObject(listRequests.get(i).getSumCredit());  
 }  
 **break**;  
 }  
 **case** 13: *// execute Credit* {  
 **int** idRequest = (Integer)inThread.readObject();  
 System.***out***.println(**"idRequest = "** + idRequest);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);  
  
 bd.changeStatus(idRequest);  
  
 **int** sumCredit = bd.getSumCodeCredit(idRequest)[0];  
 **int** codeCredit = bd.getSumCodeCredit(idRequest)[1];  
 **int** interestRate = bd.getPersentMonth(codeCredit)[0];  
 **int** period = bd.getPersentMonth(codeCredit)[1];  
  
 **int** annuity = Request.*getAnnuitySum*(sumCredit, interestRate, period);  
 **int** newSum = annuity \* period;  
 System.***out***.println(**"newSum = "** + newSum);  
  
 bd.setAnnuity(annuity, newSum, idRequest);  
 bd.addCurrentCredit(idUser, codeCredit, idRequest);  
 **break**;  
 }  
 **case** 14: *// get Requests*

{  
 **int** numberRequests = bd.getNumberRequestsDeposits();  
 System.***out***.println(**"numberRequests = "** + numberRequests);  
 outThread.writeObject(numberRequests);  
 ArrayList<Request> listRequests = bd.getRequestsDeposit();  
  
 **for** (**int** i = 0; i < numberRequests; i++) {  
 outThread.writeObject(listRequests.get(i).getIdRequest());  
 outThread.writeObject(listRequests.get(i).getId());  
 outThread.writeObject(listRequests.get(i).getCode());  
 outThread.writeObject(listRequests.get(i).getName());  
 outThread.writeObject(listRequests.get(i).getSurname());

Продолжение приложения Б

outThread.writeObject(listRequests.get(i).getLastname());  
 outThread.writeObject(listRequests.get(i).getHouseAddress());  
 outThread.writeObject(listRequests.get(i).getSumCredit());  
 }  
 **break**;  
 }  
 **case** 15: *// execute Deposit* {  
 **int** idRequest = (Integer)inThread.readObject();  
 System.***out***.println(**"idRequest = "** + idRequest);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);  
  
 **int** sumDeposit = bd.changeStatusDeposit(idRequest, idUser)[0];  
 **int** codeDeposit = bd.changeStatusDeposit(idRequest, idUser)[1];  
  
 **int** interestRate = bd.getPersentPeriod(codeDeposit)[0];  
 **int** period = bd.getPersentPeriod(codeDeposit)[1];  
  
 **int** newSum = Request.*getNewSum*(sumDeposit, interestRate, period);  
  
 bd.setNewSum(newSum, idRequest);  
 bd.addCurrentDeposit(idUser, codeDeposit, idRequest);  
 **break**;  
 }  
 **case** 16: *// get Sum For Month Credit (annuity)* {  
 System.***out***.println(**"get Sum For Month Credit"**);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);  
 **int** getSumForMonthCredit = bd.getSumForMonthCredit(idUser);  
 System.***out***.println(**"getSumForMonthCredit = "** + getSumForMonthCredit);  
 outThread.writeObject(getSumForMonthCredit);  
 **break**;  
 }  
 **case** 17: *// get Sum For All Period Credit* {  
 System.***out***.println(**"getSumForAllPeriodCredit"**);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);  
 **int** getSumForAllPeriodCredit = bd.getSumForAllPeriodCredit(idUser);  
 System.***out***.println(**"getSumForAllPeriodCredit = "** + getSumForAllPeriodCredit);  
 outThread.writeObject(getSumForAllPeriodCredit);  
 **break**;  
 }  
 **case** 18: *// cut Annuity Credit* {  
 System.***out***.println(**"cutAnnuityCredit"**);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);

Продолжение приложения Б

bd.cutAnnuityCredit(idUser);  
 **break**;  
 }  
 **case** 19: *// cut Whole Credit* {  
 System.***out***.println(**"// cut Whole Credit"**);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);  
 bd.leaveRequestWholeCredit(idUser);  
 **break**;  
 }  
 **case** 20: *// get Close Credit*

{  
 **int** numberCloseCredit = bd.getNumberCloseCredit();  
 System.***out***.println(**"numberCloseCredit = "** + numberCloseCredit);  
 outThread.writeObject(numberCloseCredit);  
 ArrayList<Request> listRequests = bd.getCloseCredit();

**for** (**int** i = 0; i < numberCloseCredit; i++) {  
 outThread.writeObject(listRequests.get(i).getIdRequest());  
 outThread.writeObject(listRequests.get(i).getId());  
 outThread.writeObject(listRequests.get(i).getCode());  
 outThread.writeObject(listRequests.get(i).getName());  
 outThread.writeObject(listRequests.get(i).getSurname());  
 outThread.writeObject(listRequests.get(i).getLastname());  
 outThread.writeObject(listRequests.get(i).getSumCredit());  
 }  
 **break**;  
 }  
 **case** 21: *// close Credit* {  
 System.***out***.println(**"closeCredit"**);  
 **int** idRequest = (Integer)inThread.readObject();  
 System.***out***.println(**"idRequest = "** + idRequest);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);  
 Request request = bd.getClosedCredit(idUser);  
 bd.closedCredit(request);  
 **break**;  
 }  
 **case** 22: *// get Close Requests Credit* {  
 **int** numberCloseRequestsCredit = bd.getNumberCloseRequestsCredit();System.***out***.println(**"numberCloseCredit = "** + numberCloseRequestsCredit);  
 outThread.writeObject(numberCloseRequestsCredit);  
 **int**[] arr = bd.getCloseRequestsCreditArr(numberCloseRequestsCredit);  
 ArrayList<Request> listRequests = bd.getCloseRequestsCredit(arr, numberCloseRequestsCredit);  
  
 **for** (**int** i = 0; i < numberCloseRequestsCredit; i++) {

Продолжение приложения Б

outThread.writeObject(listRequests.get(i).getIdRequest());

outThread.writeObject(listRequests.get(i).getId());  
 outThread.writeObject(listRequests.get(i).getCode());  
 outThread.writeObject(listRequests.get(i).getName());  
 outThread.writeObject(listRequests.get(i).getSurname());  
 outThread.writeObject(listRequests.get(i).getLastname());  
 outThread.writeObject(listRequests.get(i).getSumCredit());  
 }  
 **break**;  
 }  
 **case** 23: *// close Request Credit*

{  
 **int** idRequest = (Integer)inThread.readObject();  
 System.***out***.println(**"idRequest = "** + idRequest);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);  
 **int** sum = bd.getSumForAllPeriodCredit(idUser);  
 Request request = bd.getClosedCredit(idUser);  
 bd.closedCredit(request);  
 bd.cutMoney(sum, idUser);  
 **break**;  
 }  
 **case** 24: *// send Close Request Deposit*

{  
 System.***out***.println(**"sendCloseRequestDeposit"**);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);  
 bd.leaveRequestCloseDeposit(idUser);  
 **break**;  
 }  
 **case** 25: *// get Close Requests Deposits*

{  
 **int** numberCloseRequestsDeposits = bd.getNumberCloseRequestsDeposit();  
 System.***out***.println(**"numberCloseRequestsDeposits = "** + numberCloseRequestsDeposits);  
 outThread.writeObject(numberCloseRequestsDeposits);  
 **int**[] arr = bd.getCloseRequestsDepositArr(numberCloseRequestsDeposits);  
 **if** (numberCloseRequestsDeposits != 0) {  
  
 ArrayList<Request> listRequests = bd.getCloseRequestsDeposit(arr, numberCloseRequestsDeposits);  
  
 **for** (**int** i = 0; i < numberCloseRequestsDeposits; i++) {  
 outThread.writeObject(listRequests.get(i).getIdRequest());  
 outThread.writeObject(listRequests.get(i).getId());  
 outThread.writeObject(listRequests.get(i).getCode());  
 outThread.writeObject(listRequests.get(i).getName());  
 outThread.writeObject(listRequests.get(i).getSurname());  
 outThread.writeObject(listRequests.get(i).getLastname());

Продолжение приложения Б

outThread.writeObject(listRequests.get(i).getSumCredit());  
 }  
 }  
 **break**;  
 }  
 **case** 26: *// close Deposit* {  
 System.***out***.println(**"closeDeposit"**);  
 **int** idRequest = (Integer)inThread.readObject();  
 System.***out***.println(**"idRequest = "** + idRequest);  
 **int** idUser = (Integer)inThread.readObject();  
 System.***out***.println(**"idUser = "** + idUser);  
 **int** sum = bd.getSumDeposit(idUser);  
 Request request = bd.getClosedDeposit(idUser);  
 bd.closedDeposit(request);  
 bd.addMoney(idUser, sum);  
 **break**;  
 }**case** 27: *// getTariffsCodes* {  
 System.***out***.println(**"// analysis Popularity Credits"**);  
 **int**[] tariffsCodes = bd.getTariffsCodes();  
 outThread.writeObject(tariffsCodes.**length**);  
 **for** (**int** i = 0; i < tariffsCodes.**length**; i++) {  
 outThread.writeObject(tariffsCodes[i]);  
 System.***out***.println(tariffsCodes[i]);  
 }  
 **break**;  
 }  
 **case** 28: *// getCountCredits* {  
 **int**[] requests = bd.getCountCredits(bd.getTariffsCodes());  
 outThread.writeObject(requests.**length**);  
 **for** (**int** i = 0; i < requests.**length**; i++) {  
 outThread.writeObject(requests[i]);  
 System.***out***.println(requests[i]);  
 }  
 **break**;  
 }  
 **case** 29: *// deleteCredit* {  
 **int** codeCredit = (Integer)inThread.readObject();  
 System.***out***.println(**"codeCredit = "** + codeCredit);  
 bd.deleteCredit(codeCredit);  
 **break**;  
 }  
 **case** 30: *// sendAddCredit* {  
 String title = (String)inThread.readObject();  
 System.***out***.println(**"title = "** + title);  
 String loanPurpose = (String)inThread.readObject();  
 System.***out***.println(**"loanPurpose = "** + loanPurpose);  
 **int** min = (Integer)inThread.readObject();  
 System.***out***.println(**"min = "** + min);  
 **int** max = (Integer)inThread.readObject();  
 System.***out***.println(**"max = "** + max);  
 **int** period = (Integer)inThread.readObject();  
 System.***out***.println(**"period = "** + period);  
 **int** inrerestRate = (Integer)inThread.readObject();  
 System.***out***.println(**"inrerestRate = "** + inrerestRate);

Продолжение приложения Б

**int** warranters = (Integer)inThread.readObject();  
 System.***out***.println(**"warranters = "** + warranters);  
 Tariff tariff = **new** Tariff(title, loanPurpose, min, max, period, inrerestRate, warranters);  
 bd.addNewCredit(tariff);  
 **break**;  
 }  
 **case** 31: *// getTariffsCodesDeposit* {  
 System.***out***.println(**"getTariffsCodesDeposit"**);  
 **int**[] tariffsCodes = bd.getTariffsCodesDeposit();  
 outThread.writeObject(tariffsCodes.**length**);  
 **for** (**int** i = 0; i < tariffsCodes.**length**; i++) {  
 outThread.writeObject(tariffsCodes[i]);  
 System.***out***.println(tariffsCodes[i]);  
 }  
 **break**;  
 }  
 **case** 32: *// getCountDeposits* {  
 **int**[] requests = bd.getCountDeposits(bd.getTariffsCodesDeposit());  
 outThread.writeObject(requests.**length**);  
 **for** (**int** i = 0; i < requests.**length**; i++) {  
 outThread.writeObject(requests[i]);  
 System.***out***.println(requests[i]);  
 }  
 **break**;  
 }  
 **case** 33: *// sendAddCredit* {  
 String title = (String)inThread.readObject();  
 System.***out***.println(**"title = "** + title);  
 **int** min = (Integer)inThread.readObject();  
 System.***out***.println(**"min = "** + min);  
 **int** period = (Integer)inThread.readObject();  
 System.***out***.println(**"period = "** + period);  
 **int** inrerestRate = (Integer)inThread.readObject();  
 System.***out***.println(**"inrerestRate = "** + inrerestRate);  
 Deposit deposit = **new** Deposit(title, min, period, inrerestRate);  
 bd.addNewDeposit(deposit);  
 **break**;  
 }  
 **case** 34: *// deleteDeposit* {  
 **int** codeDeposit = (Integer)inThread.readObject();  
 System.***out***.println(**"codeDeposit = "** + codeDeposit);  
 bd.deleteDeposit(codeDeposit);  
 **break**;  
 }  
 **case** 35: *// getAnyErrorsAuthorization* {  
 String login = (String)inThread.readObject();  
 System.***out***.println(**"login "** + login);  
 String password = (String)inThread.readObject();  
 System.***out***.println(**"password "** + password);  
 **int** success = bd.getAnyErrorsAuthorization(login, password);  
 outThread.writeObject(success);  
 **break**;

Продолжение приложения Б

}  
 **case** 36: *// getOpenCredits // получение открытых кредитов* {  
 **int** numberOpenCredit = bd.getNumberOpenCredit();  
 System.***out***.println(**"numberOpenCredit = "** + numberOpenCredit);  
 outThread.writeObject(numberOpenCredit);  
 ArrayList<Request> listRequests = bd.getOpenCredit(); *// получение открытых кредитов* **for** (**int** i = 0; i < numberOpenCredit; i++) {  
 outThread.writeObject(listRequests.get(i).getIdRequest());  
 outThread.writeObject(listRequests.get(i).getId());  
 outThread.writeObject(listRequests.get(i).getCode());  
 outThread.writeObject(listRequests.get(i).getName());  
 outThread.writeObject(listRequests.get(i).getSurname());  
 outThread.writeObject(listRequests.get(i).getLastname());  
 outThread.writeObject(listRequests.get(i).getHouseAddress());  
 outThread.writeObject(listRequests.get(i).getJobAddress());  
 outThread.writeObject(listRequests.get(i).getSalary());  
 outThread.writeObject(listRequests.get(i).getSumCredit());  
 }  
 **break**;  
 }  
 **case** 37: *// getWaitCredits // получение ожидающих кредитов* {  
 **int** numberWaitCredit = bd.getNumberWaitCredit();  
 System.***out***.println(**"numberWaitCredit = "** + numberWaitCredit);  
 outThread.writeObject(numberWaitCredit);  
 ArrayList<Request> listRequests = bd.getWaitCredit(); *// получение ожидающих кредитов* **for** (**int** i = 0; i < numberWaitCredit; i++) {  
 outThread.writeObject(listRequests.get(i).getIdRequest());  
 outThread.writeObject(listRequests.get(i).getId());  
 outThread.writeObject(listRequests.get(i).getCode());  
 outThread.writeObject(listRequests.get(i).getName());  
 outThread.writeObject(listRequests.get(i).getSurname());  
 outThread.writeObject(listRequests.get(i).getLastname());  
 outThread.writeObject(listRequests.get(i).getHouseAddress());  
 outThread.writeObject(listRequests.get(i).getJobAddress());

Продолжение приложения Б

outThread.writeObject(listRequests.get(i).getSalary());  
 outThread.writeObject(listRequests.get(i).getSumCredit());  
 }  
 **break**;  
 }  
 **case** 38: *// getTurnOffCredits // получение закрытых кредитов* {  
 **int** numberTurnOffCredit = bd.getNumberTurnOffCredit();  
 System.***out***.println(**"numberTurnOffCredit = "** + numberTurnOffCredit);  
 outThread.writeObject(numberTurnOffCredit);  
 ArrayList<Request> listRequests = bd.getTurnOffCredit(); *// получение ожидающих кредитов* **for** (**int** i = 0; i < numberTurnOffCredit; i++) {  
 outThread.writeObject(listRequests.get(i).getIdRequest());  
 outThread.writeObject(listRequests.get(i).getId());  
 outThread.writeObject(listRequests.get(i).getCode());  
 outThread.writeObject(listRequests.get(i).getName());  
 outThread.writeObject(listRequests.get(i).getSurname());  
 outThread.writeObject(listRequests.get(i).getLastname());  
 outThread.writeObject(listRequests.get(i).getHouseAddress());  
 outThread.writeObject(listRequests.get(i).getJobAddress());  
 outThread.writeObject(listRequests.get(i).getSalary());  
 outThread.writeObject(listRequests.get(i).getSumCredit());  
 }  
 **break**;  
 }  
 **case** 39: *// getOpenDeposits // получение открытых депозитов* {  
 **int** numberOpenDeposit = bd.getNumberOpenDeposit();  
 System.***out***.println(**"numberOpenDeposit = "** + numberOpenDeposit);  
 outThread.writeObject(numberOpenDeposit);  
 ArrayList<Request> listRequests = bd.getOpenDeposit(); *// получение открытых кредитов* **for** (**int** i = 0; i < numberOpenDeposit; i++) {  
 outThread.writeObject(listRequests.get(i).getIdRequest());  
 outThread.writeObject(listRequests.get(i).getId());  
 outThread.writeObject(listRequests.get(i).getCode());  
 outThread.writeObject(listRequests.get(i).getName());

Продолжение приложения Б

outThread.writeObject(listRequests.get(i).getSurname());  
 outThread.writeObject(listRequests.get(i).getLastname());  
 outThread.writeObject(listRequests.get(i).getHouseAddress());  
 outThread.writeObject(listRequests.get(i).getSumCredit());  
 }  
 **break**;  
 }  
 **case** 40: *// getWaitDeposits // получение ожидающих депозитов* {  
 **int** numberWaitDeposit = bd.getNumberWaitDeposit();  
 System.***out***.println(**"numberWaitDeposit = "** + numberWaitDeposit);  
 outThread.writeObject(numberWaitDeposit);  
 ArrayList<Request> listRequests = bd.getWaitDeposit(); *// получение ожидающих кредитов* **for** (**int** i = 0; i < numberWaitDeposit; i++) {  
 outThread.writeObject(listRequests.get(i).getIdRequest());  
 outThread.writeObject(listRequests.get(i).getId());  
 outThread.writeObject(listRequests.get(i).getCode());  
 outThread.writeObject(listRequests.get(i).getName());  
 outThread.writeObject(listRequests.get(i).getSurname());  
 outThread.writeObject(listRequests.get(i).getLastname());  
 outThread.writeObject(listRequests.get(i).getHouseAddress());  
 outThread.writeObject(listRequests.get(i).getSumCredit());  
 }  
 **break**;  
 }  
 **case** 41: *// getTurnOffDeposits // получение закрытых ддепозитов* {  
 **int** numberTurnOffDeposit = bd.getNumberTurnOffDeposit();  
 System.***out***.println(**"numberTurnOffDeposit = "** + numberTurnOffDeposit);  
 outThread.writeObject(numberTurnOffDeposit);  
 ArrayList<Request> listRequests = bd.getTurnOffDeposit(); *// получение ожидающих кредитов* **for** (**int** i = 0; i < numberTurnOffDeposit; i++) {  
 outThread.writeObject(listRequests.get(i).getIdRequest());  
 outThread.writeObject(listRequests.get(i).getId());  
 outThread.writeObject(listRequests.get(i).getCode());  
 outThread.writeObject(listRequests.get(i).getName());

Продолжение приложения Б

outThread.writeObject(listRequests.get(i).getSurname());  
 outThread.writeObject(listRequests.get(i).getLastname());  
 outThread.writeObject(listRequests.get(i).getHouseAddress());  
 outThread.writeObject(listRequests.get(i).getSumCredit());  
 }  
 **break**;  
 }  
 **case** 99999:  
 flag = **false**;  
 **break**;  
 }  
 System.***out***.println(**"exit "** + flag);  
 }  
  
 }**catch** (Exception e) {  
 e.printStackTrace();  
 System.***out***.println(**"Client: "** + e.getMessage());  
 }**finally** {  
 **try** {  
 inThread.close();  
 outThread.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

# **ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Листинг скрипта генерации базы данных**

*-- -----------------------------------------------------  
-- Schema db\_bank  
-- -----------------------------------------------------***CREATE SCHEMA IF NOT EXISTS** `db\_bank` **DEFAULT CHARACTER SET** *utf8* ;  
**USE** `db\_bank` ;  
  
*-- -----------------------------------------------------  
-- Table `db\_bank`.`users`  
-- -----------------------------------------------------***CREATE TABLE IF NOT EXISTS** `db\_bank`.`users` (  
 **`id` INT**(11) **NOT NULL AUTO\_INCREMENT**,  
 **`login` VARCHAR**(20) **NOT NULL**,  
 **`password` VARCHAR**(25) **NOT NULL**,  
 **`access` INT**(1) **NOT NULL**,  
 **PRIMARY KEY** (**`id`**))  
**ENGINE** = InnoDB  
**AUTO\_INCREMENT** = 37  
**DEFAULT CHARACTER SET** = *utf8*;  
  
**CREATE UNIQUE INDEX** `id\_UNIQUE` **ON** `db\_bank`.`users` (**`id` ASC**);  
  
**CREATE UNIQUE INDEX** `login\_UNIQUE` **ON** `db\_bank`.`users` (**`login` ASC**);  
  
  
*-- -----------------------------------------------------  
-- Table `db\_bank`.`creditsrequests`  
-- -----------------------------------------------------***CREATE TABLE IF NOT EXISTS** `db\_bank`.`creditsrequests` (  
 **`idRequest` INT**(11) **NOT NULL AUTO\_INCREMENT**,  
 **`idUser` INT**(11) **NOT NULL**,  
 **`codeCredit` INT**(11) **NOT NULL**,  
 **`name` VARCHAR**(45) **NOT NULL**,  
 **`surname` VARCHAR**(45) **NOT NULL**,  
 **`lastname` VARCHAR**(45) **NOT NULL**,  
 **`houseAddress` VARCHAR**(60) **NOT NULL**,  
 **`jobAddress` VARCHAR**(60) **NOT NULL**,  
 **`salary` INT**(11) **NOT NULL**,  
 **`sumCredit` INT**(11) **NOT NULL**,  
 **`annuity` INT**(11) **NULL DEFAULT NULL**,  
 **`status` INT**(11) **NULL DEFAULT NULL**,  
 **PRIMARY KEY** (**`idRequest`**, **`idUser`**))  
**ENGINE** = InnoDB  
**AUTO\_INCREMENT** = 14  
**DEFAULT CHARACTER SET** = *utf8*;  
  
**CREATE UNIQUE INDEX** `idRequest\_UNIQUE` **ON** `db\_bank`.`creditsrequests` (**`idRequest` ASC**);  
  
  
*-- -----------------------------------------------------  
-- Table `db\_bank`.`clients\_credit\_and\_request`  
-- -----------------------------------------------------***CREATE TABLE IF NOT EXISTS** `db\_bank`.`clients\_credit\_and\_request` (  
 **`idUser` INT**(11) **NOT NULL**,  
 **`codeCredit` INT**(11) **NOT NULL**,  
 **`idRequest` INT**(11) **NOT NULL**,  
 **`close` INT**(11) **NULL DEFAULT NULL**,  
 **PRIMARY KEY** (**`idUser`**, **`idRequest`**))

Продолжение приложения В

**ENGINE** = InnoDB  
**DEFAULT CHARACTER SET** = *utf8*;  
  
**CREATE UNIQUE INDEX** `idUser\_UNIQUE` **ON** `db\_bank`.`clients\_credit\_and\_request` (**`idUser` ASC**);  
  
  
*-- -----------------------------------------------------  
-- Table `db\_bank`.`depositsrequests`  
-- -----------------------------------------------------***CREATE TABLE IF NOT EXISTS** `db\_bank`.`depositsrequests` (  
 **`idRequest` INT**(11) **NOT NULL AUTO\_INCREMENT**,  
 **`idUser` INT**(11) **NOT NULL**,  
 **`codeDeposit` INT**(11) **NOT NULL**,  
 **`name` VARCHAR**(45) **NOT NULL**,  
 **`surname` VARCHAR**(45) **NOT NULL**,  
 **`lastname` VARCHAR**(45) **NOT NULL**,  
 **`houseAddress` VARCHAR**(45) **NOT NULL**,  
 **`sumDeposit` INT**(11) **NOT NULL**,  
 **`status` INT**(11) **NULL DEFAULT NULL**,  
 **PRIMARY KEY** (**`idRequest`**))  
**ENGINE** = InnoDB  
**AUTO\_INCREMENT** = 7  
**DEFAULT CHARACTER SET** = *utf8*;  
  
**CREATE UNIQUE INDEX** `idRequest\_UNIQUE` **ON** `db\_bank`.`depositsrequests` (**`idRequest` ASC**);  
  
  
*-- -----------------------------------------------------  
-- Table `db\_bank`.`clients\_deposit\_and\_request`  
-- -----------------------------------------------------***CREATE TABLE IF NOT EXISTS** `db\_bank`.`clients\_deposit\_and\_request` (  
 **`idUser` INT**(11) **NOT NULL**,  
 **`codeDeposit` INT**(11) **NOT NULL**,  
 **`idRequest` INT**(11) **NOT NULL**,  
 **`close` INT**(11) **NULL DEFAULT NULL**,  
 **PRIMARY KEY** (**`idUser`**, **`idRequest`**))  
**ENGINE** = InnoDB  
**DEFAULT CHARACTER SET** = *utf8*;  
  
**CREATE UNIQUE INDEX** `idUser\_UNIQUE` **ON** `db\_bank`.`clients\_deposit\_and\_request` (**`idUser` ASC**);  
  
  
*-- -----------------------------------------------------  
-- Table `db\_bank`.`closedcredits`  
-- -----------------------------------------------------***CREATE TABLE IF NOT EXISTS** `db\_bank`.`closedcredits` (  
 **`idRequest` INT**(11) **NOT NULL**,  
 **`idUser` INT**(11) **NOT NULL**,  
 **`codeCredit` INT**(11) **NOT NULL**,  
 **`name` VARCHAR**(45) **NOT NULL**,  
 **`surname` VARCHAR**(45) **NOT NULL**,  
 **`lastname` VARCHAR**(45) **NOT NULL**,  
 **PRIMARY KEY** (**`idRequest`**))  
**ENGINE** = InnoDB  
**DEFAULT CHARACTER SET** = *utf8*;  
  
**CREATE UNIQUE INDEX** `idRequest\_UNIQUE` **ON** `db\_bank`.`closedcredits` (**`idRequest` ASC**);

Продолжение приложения В

*-- -----------------------------------------------------  
-- Table `db\_bank`.`closeddeposits`  
-- -----------------------------------------------------***CREATE TABLE IF NOT EXISTS** `db\_bank`.`closeddeposits` (  
 **`idRequest` INT**(11) **NOT NULL**,  
 **`idUser` INT**(11) **NOT NULL**,  
 **`codeDeposit` INT**(11) **NOT NULL**,  
 **`name` VARCHAR**(45) **NOT NULL**,  
 **`surname` VARCHAR**(45) **NOT NULL**,  
 **`lastname` VARCHAR**(45) **NOT NULL**,  
 **PRIMARY KEY** (**`idRequest`**))  
**ENGINE** = InnoDB  
**DEFAULT CHARACTER SET** = *utf8*;  
  
**CREATE UNIQUE INDEX** `idRequest\_UNIQUE` **ON** `db\_bank`.`closeddeposits` (**`idRequest` ASC**);  
  
  
*-- -----------------------------------------------------  
-- Table `db\_bank`.`deposits`  
-- -----------------------------------------------------***CREATE TABLE IF NOT EXISTS** `db\_bank`.`deposits` (  
 **`code` INT**(11) **NOT NULL AUTO\_INCREMENT**,  
 **`title` VARCHAR**(45) **NOT NULL**,  
 **`minSum` INT**(11) **NOT NULL**,  
 **`period` INT**(11) **NOT NULL**,  
 **`interestRate` INT**(11) **NOT NULL**,  
 **PRIMARY KEY** (**`code`**))  
**ENGINE** = InnoDB  
**AUTO\_INCREMENT** = 9  
**DEFAULT CHARACTER SET** = *utf8*;  
  
**CREATE UNIQUE INDEX** `code\_UNIQUE` **ON** `db\_bank`.`deposits` (**`code` ASC**);  
  
  
*-- -----------------------------------------------------  
-- Table `db\_bank`.`moneybag`  
-- -----------------------------------------------------***CREATE TABLE IF NOT EXISTS** `db\_bank`.`moneybag` (  
 **`id` INT**(11) **NOT NULL**,  
 **`grossAssets` INT**(11) **NULL DEFAULT NULL**,  
 **PRIMARY KEY** (**`id`**))  
**ENGINE** = InnoDB  
**DEFAULT CHARACTER SET** = *utf8*;  
  
**CREATE UNIQUE INDEX** `id\_UNIQUE` **ON** `db\_bank`.`moneybag` (**`id` ASC**);  
  
  
*-- -----------------------------------------------------  
-- Table `db\_bank`.`tariffs`  
-- -----------------------------------------------------***CREATE TABLE IF NOT EXISTS** `db\_bank`.`tariffs` (  
 **`code` INT**(11) **NOT NULL AUTO\_INCREMENT**,  
 **`title` VARCHAR**(45) **NOT NULL DEFAULT 'Credit'**,  
 **`loanPurpose` VARCHAR**(60) **NOT NULL**,  
 **`minimumAmount` INT**(11) **NOT NULL**,  
 **`maximumAmount` INT**(11) **NOT NULL**,  
 **`period` INT**(11) **NOT NULL**,  
 **`interestRate` INT**(11) **NOT NULL**,  
 **`warranters` INT**(11) **NOT NULL**,  
 **PRIMARY KEY** (**`code`**))

Продолжение приложения В

**ENGINE** = InnoDB  
**AUTO\_INCREMENT** = 16  
**DEFAULT CHARACTER SET** = *utf8*;  
  
**CREATE UNIQUE INDEX** `code\_UNIQUE` **ON** `db\_bank`.`tariffs` (**`code` ASC**);  
  
  
*-- -----------------------------------------------------  
-- Administrator  
-- -----------------------------------------------------***INSERT INTO** `db\_bank`.`users` (**login**, **password**, **access**) **values**(**'admin'**, **'admin'**, 0);