Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Контрольная работа №1 по ТРПО

«Учет и анализ банковских операций»

Выполнил: Нагорный Евгений Олегович

студент 2 курса, специальность ПОИТ

группа № 581072

Домашний адрес: ул. В. Хоружей 19, кв. 44

Телефон: +375297583548

Электронный адрес: jenia010696@mail.ru

Проверила: Бакунова Оксана Михайловна

Минск 2017

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc472441806)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc472441807)

[1.1 Постановка задачи 4](#_Toc472441808)

[1.2 Описание и обоснование выбранного языка программирования 5](#_Toc472441809)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 7](#_Toc472441810)

[3 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ 11](#_Toc472441811)

[3.1 Диаграмма вариантов использования 11](#_Toc472441812)

[3.2 Диаграмма классов 12](#_Toc472441813)

[3.3 Диаграмма деятельности 12](#_Toc472441814)

[3.4 Диаграмма коопераций 13](#_Toc472441815)

[3.5 Диаграмма последовательности 14](#_Toc472441816)

[3.6 Диаграмма развёртывания 14](#_Toc472441817)

[3.7 Диаграмма состояний 15](#_Toc472441818)

[3.8 Диаграмма компонентов 16](#_Toc472441819)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 17](#_Toc472441820)

[Генерация кода 18](#_Toc472441821)

[Фрагмент кода программы 21](#_Toc472441822)

# ВВЕДЕНИЕ

Банковский сектор – одно из важнейших направлений развития рыночных отношений, который является основой для нормального, эффективного функционирования рыночного механизма.

Коммерческий банк в становится основным элементом банковской системы. Именно поэтому развитие данного направления должно быть приоритетным, потому что действие кредитно-финансового механизма и определяет уровень развития экономики страны в целом.

В рыночной экономике банк выполняет свою главную функцию –посредничество в кредите, которое он осуществляет путем перераспределения денежных средств, временно высвобождающихся в процессе кругооборота фондов предприятий и денежных доходов частных лиц. Коммерческие банки выполняют роль посредников между хозяйственными единицами и секторами, накапливающими временно свободные денежные средства, и теми участниками экономического оборота, которые временно нуждаются в дополнительном капитале.

Кредитные операции банков и кредитных учреждений при первом приближении делятся на активные и пассивные. В первом случае банк является кредитором, т.е. стороной, дающей кредит (ссужающей деньги) и размещающей денежные средства в форме депозитов (вкладов), во втором – дебитором, т.е. стороной, берущей кредит и принимающей денежные средства в форме депозитов (вкладов). Т. е. и активные, и пассивные кредитные операции предполагают использование как кредитов (ссуд), так и депозитов (вкладов).

Такое ПС должно позволять:

- вводить и хранить информацию о получателях кредитов/депозитов, о самих кредитах/депозитах, целях выдачи кредитов/депозитов банком и т.п.;

- обрабатывать хранимую информацию определенным образом (поиск, сортировка) в соответствии со специфическими задачами предметной области.

Разрабатываемая система должна быть построена согласно принципам архитектуры «клиент-сервер», что позволяет существенно упростить ее модификацию, развертывание и переносимость.

Целью работы является разработка система учёта и анализа банковских операций по депозитам и кредитам для физических лиц. Программа должна быть организована согласно принципам архитектуры «клиент-сервер», клиент должен взаимодействовать с сервером по протоколу TCP/IP. Данные должны храниться посредством СУБД Sybase 9.0.

# 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

# 1.1 Постановка задачи

Человек, желающий взять кредит/депозит, получает его под определенные нужды или кладет с определенными целями. Цель получения кредита характеризует вид кредита – на покупку автомобиля, на потребительские нужды и т.п. Типы вкладов подразделяясь по срокам служат для получения краткосрочных процентов или долгосрочные сберегательные вклады и т.п.

При получении кредита банк учитывает паспортные данные получателя кредита: серию и номер паспорта, адрес прописки и контактный телефон. Однажды учтенная, эта информация позволяет в будущем проследить кредитную историю получателя кредитов – какие кредиты он брал раньше, насколько исправно он их погашал и т.п. Результаты проверки кредитной истории могут оказывать влияние на принятие решения о выдаче кредита.

При получении кредита требуются определенные документы, характеризующие личность получателя кредита. Это, как правило, его паспорт и справка с места работы с указанием заработной платы за последние 3 месяца. Размер заработной платы влияет на сумму кредита, которая может быть выдана банком.

Лицо, желающее взять кредит, приходит в банк и предоставляет ему все требуемые бумаги. Служащие банка проверяют достоверность указанных в этих бумагах сведений, определяют возможность выдачи кредита в зависимости от кредитной истории получателя кредита, определяют максимальную сумму кредита, которая может быть выдана в зависимости от размера з/п., определяют срок погашения кредита и проценты по кредиту.

Для открытия счета не требуются столь сложные процедуры и не выставляются такие требования к клиенту.

Следует заметить, что кредит может выдаваться в различной валюте, так же как и открытие депозита. Для РБ кредиты могут выдаваться в белорусских рублях, евро и долларах США.

После выдачи кредита служащий каким-либо образом отмечает факт выдачи кредита. Впоследствии данная информация может быть использована для анализа кредитной истории получателя кредита.

Учет сведений о полученном кредите соответствует добавлению записи в разрабатываемом ПС.

Анализ предметной области позволил выявить минимальный набор из 8-х сущностей. Это будут:

1. Валюта – информация о валюте, в которой предоставляется кредит.
2. Клиент – информация о физическом лице – получателе кредита/депозита.
3. Кредит – выдаваемые кредиты банком.
4. Депозит – открываемые типы вкладов.
5. Заключение кредитного договора – данные о дате и сумме заключения договора о выдаче кредита.
6. Оформление вклада – данные о дате и размере вклада открываемого депозита.
7. Выплаты по кредитам – данные о взносах по погашению кредитов.
8. Пополнения депозита – данные о новых пополнениях депозита со стороны клиента.

# 1.2 Описание и обоснование выбранного языка программирования

Java - объектно-ориентированный язык программирования, разрабатываемый компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 года. Изначально новый язык программирования назывался Oak (James Gosling) и разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения

Отличительной особенностью Java в сравнении с другими языками программирования общего назначения является обеспечение высокой продуктивности программирования, нежели производительность работы приложения или эффективность использования им памяти.

В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. В Java добавлены все хорошие черты C++.

Три ключевых элемента объединились в технологии языка Java:

1) Java предоставляет для широкого использования свои апплеты (applets) — небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML.

2) Java высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты.

3) Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов.

Огромное преимущество Java заключается в том, что на этом языке можно создавать приложения, способные работать на различных платформах. К сети Internet подключены компьютеры самых разных типов - Pentium PC, Macintosh, рабочие станции Sun и так далее. Даже в рамках компьютеров, созданных на базе процессоров Intel, существует несколько платформ, например, Microsoft Windows версии 3.1, Windows 95, Windows NT, OS/2, Solaris, различные разновидности операционной системы UNIX с графической оболочкой X­Windows. Между тем, создавая сервер Web в сети Internet, хотелось бы, чтобы им могло пользоваться как можно большее число людей. В этом случае выручат приложения Java, предназначенные для работы на различных платформах и не зависящие от конкретного типа процессора и операционной системы.

Программы, составленные на языке программирования Java, можно разделить по своему назначению на две большие группы.

К первой группе относятся приложения Java, предназначенные для автономной работы под управлением специальной интерпретирующей машины Java. Реализации этой машины созданы для всех основных компьютерных платформ.

Вторая группа - это так называемые аплеты (applets). Аплеты представляют собой разновидность приложений Java, которые интерпретируются виртуальной машиной Java, встроенной практически во все современные браузеры.

Приложения, относящиеся к первой группе - это обычные автономные программы. Так как они не содержат машинного кода и работают под управлением специального интерпретатора, их производительность заметно ниже, чем у обычных программ, составленных, например, на языке программирования C++. Программы Java без перетрансляции способны работать на любой платформе, что само по себе имеет большое значение в плане разработок для Internet.

Аплеты Java встраиваются в документы HTML, хранящиеся на сервере Web. С помощью аплетов вы можете сделать страницы сервера Web динамичными и интерактивными. Аплеты позволяют выполнять сложную локальную обработку данных, полученных от сервера Web или введенных пользователем с клавиатуры. Из соображений безопасности аплеты (в отличие от обычных приложений Java) не имеют никакого доступа к файловой системе локального компьютера. Все данные для обработки они могут получить только от сервера Web. Более сложную обработку данных можно выполнять, организовав взаимодействие между аплетами и расширениями сервера Web - приложениями CGI и ISAPI.

Для повышения производительности приложений Java в современных браузерах используется компиляция "на лету"- Just-In-Time compilation (JIT). При первой загрузке аплета его код транслируется в обычную исполнимую программу, которая сохраняется на диске и запускается. В результате общая скорость выполнения аплета Java увеличивается в несколько раз.

Язык Java является объектно-ориентированным и поставляется с достаточно объемной библиотекой классов. Так же как и библиотеки классов систем разработки приложений на языке C++, библиотеки классов Java значительно упрощают разработку приложений, представляя в распоряжение программиста мощные средства решения распространенных задач. Поэтому программист может больше внимания уделить решению прикладных задач.

Язык Java специально ориентирован на самые передовые технологии, связанные с сетью Internet. Растущая популярность Internet и, в особенности, серверов Web, создает для программистов новые возможности для реализации своих способностей.

# 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В работе проводится моделирование с использование IDEF0(BPWin), UML (Rational Rose 2000), IDEF1x (ErWin).

1. Важная роль отводится процессу функционального проектирования.

Для регламентирования создания функциональных моделей ПС предназначен стандарт IDEF0 (Integrated Definition Function Modeling), который и реализован в пакете BpWin.

В основе IDEF0 лежит понятие блока, который реализует некую конкретную функцию. Четыре стороны блока имеют разное назначение. Слева отображаются входные данные (исходные данные). Справа – выходные данные (результат выполнения функции). Сверху – управление (управляющие воздействия на функцию). Снизу – механизм (посредством чего реализуется данная функция).

Функция – это управляемое действие над входными данными, результатом которого являются выходные данные, при этом используется некий механизм. Взаимодействие между функциями отображается в виде стрелок. Иногда стороны блока называют направлениями, а стрелки потоками. Стрелки можно подписывать. Подписи связываются с конкретной стрелкой при помощи зигзага.

В основе IDEF0 лежит три базовых принципа:

1) принцип функциональной декомпозиции – любая функция может быть разбита (декомпозирована) на более простые функции (более понятен термин детализация);

2) принцип ограничения сложности – количество блоков на диаграмме должно быть не менее двух, но не более шести (условие удобочитаемости);

3) принцип контекста – моделирование делового процесса начинается с построения контекстной диаграммы, на которой отображается только один блок – главная функция моделирующей системы, ограничивающая область границы моделирующей системы (регламентирует начальный этап построения модели).

Под субъектом понимается сама система, при этом необходимо точно установить, что входит в систему, а что является внешним воздействием на систему. Т. е. первоначально нужно определить область моделирования. Описание области как системы в целом, так и ее компонентов является основой построения модели.

IDEF1X является методом для разработки реляционных баз данных и использует условный синтаксис, специально разработанный для удобного построения концептуальной схемы. Концептуальной схемой мы называем универсальное представление структуры данных в рамках коммерческого предприятия, независимое от конечной реализации базы данных и аппаратной платформы. Будучи статическим методом разработки, IDEF1X изначально не предназначен для динамического анализа по принципу "AS IS", тем не менее, он иногда применяется в этом качестве, как альтернатива методу IDEF1. Использование метода IDEF1X наиболее целесообразно для построения логической структуры базы данных после того, как все информационные ресурсы исследованы (скажем с помощью метода IDEF1) и решение о внедрении реляционной базы данных, как части корпоративной информационной системы, было принято.

Хотя терминология IDEF1X практически совпадает с терминологией IDEF1, существует ряд фундаментальных отличий в теоретических концепциях этих методологий. Сущность в IDEF1X описывает собой совокупность или набор экземпляров похожих по свойствам, но однозначно отличаемых друх от друга по одному или нескольким признакам. Каждый экземпляр является реализацией сущности. Таким образом, сущность в IDEF1X описывает конкретный набор экземпляров реального мира, в отличие от сущности в IDEF1, которая представляет собой абстрактный набор информационных отображений реального мира. Примером сущности IDEF1X может быть сущность "СОТРУДНИК", которая представляет собой всех сотрудников предприятия, а один из них, скажем, Иванов Петр Сергеевич, является конкретной реализацией этой сущности.

Сase-средство BPwin значительно облегчает задачу создания информационной системы, позволяя осуществить декомпозицию сложной системы на более простые с тем, чтобы каждая из них могла проектироваться независимо, и для понимания любого уровня проектирования достаточно было оперировать с информацией о немногих ее частях.

Рассмотрим процесс моделирования в методологии IDEF0 на примере контекстной диаграммы разработанной модели процесса учета кредитных операций в банке.

Цель модели – моделирование процесса учета выданных кредитов.

Точка зрения модели – банковский служащий, отвечающий за выдачу кредитов.

Входы модели (диаграммы):

1. Паспортные данные получателя кредита – содержит информацию о получателе кредита;
2. Кредитная история – сведения о лицах, которые уже брали ранее у банка кредита для определения степени благонадежности получателя кредита;
3. Пожелания получателя кредита – определяют цель кредитования, размер кредита, сроки погашения и т.п.;
4. Время – все этапы процесса учета требуют затрат времени.

Процесс учета регламентируется порядком выдачи кредита и порядком учета кредитных операций в банке.

Выходные процессы учета – запись об учтенной кредитной операции, информация об ошибке и отчетная документация.

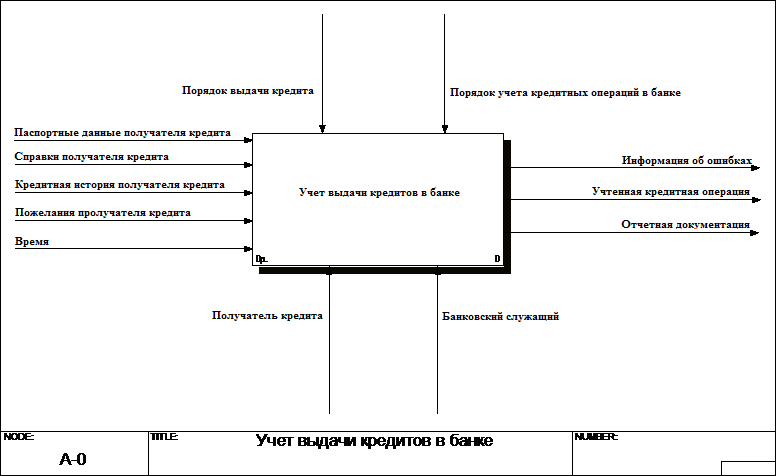


Рисунок 2.1- Контекстная диаграмма модели

Декомпозиция первого уровня контекстной диаграммы делится на 4 составных блока:

- определение благонадежности получателя кредита;

- определение возможности выдачи кредита;

- учет информации о кредитополучателе;

- учет информации о выданном кредите.

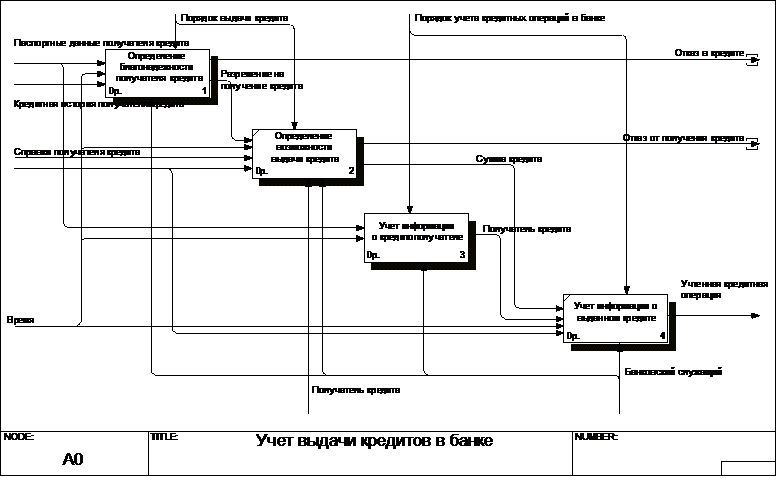


Рисунок 2.2 – Декомпозиция контекстной диаграммы функциональной модели

После построения диаграммы декомпозиции первого уровня для указанных на ней функций строятся отдельные диаграммы (диаграммы декомпозиции второго уровня). Затем процесс декомпозиции (построения диаграмм) продолжается до тех пор, пока дальнейшая детализация функций не теряет смысла. Для каждой атомарной функции, описывающей элементарную операцию (т. е. функции, не имеющей диаграмму декомпозиции), составляется подробная спецификация, определяющая ее особенности и алгоритм реализации. В качестве дополнения к спецификации могут использоваться блок-схемы алгоритмов. Таким образом, процесс функционального моделирования заключается в постепенном выстраивании иерархии функций.

# 3 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ

Основное назначение логического представления состоит в анализе структурных и функциональных отношений между элементами модели системы. Различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения.

# 3.1 Диаграмма вариантов использования

Модель вариантов использования предназначается для определения требований к системе. Она включает в себя актеров, варианты использования и связи между ними.

Разработка диаграммы вариантов использования преследует цели:

* определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
* сформулировать общие требования к функциональному поведению проектируемой системы;
* разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
* подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

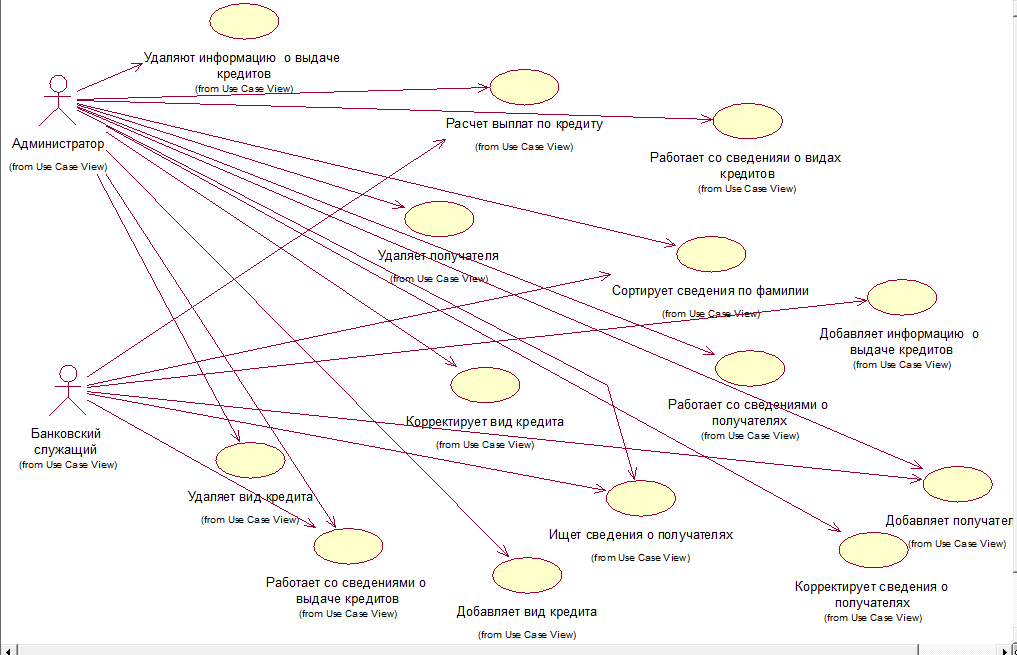


Рисунок 3.1 – Диаграмма вариантов использования

# 3.2 Диаграмма классов

Диаграмма классов - диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними. Входит в UML.

Существует два вида:

* Статический вид диаграммы рассматривает логические взаимосвязи классов между собой;
* Аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов, входящих в систему.

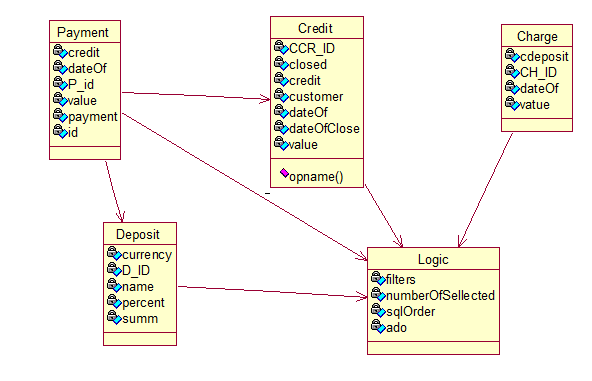
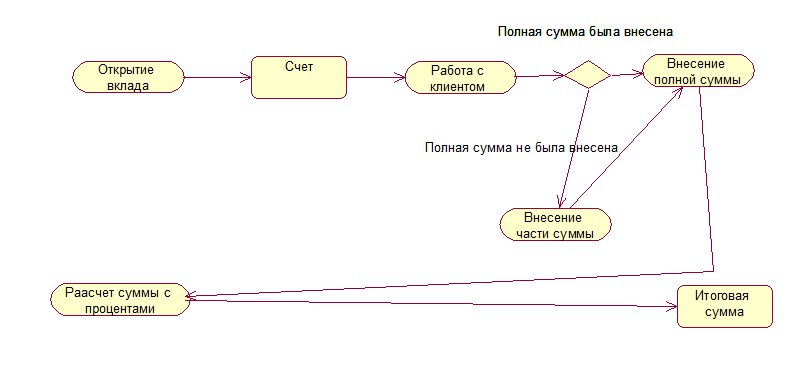


Рисунок 3.2 – Диаграмма классов

# 3.3 Диаграмма деятельности

Диаграмма деятельности — UML-диаграмма, на которой показано разложение некоторой деятельности на её составные части. Под деятельностью (англ. activity) понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий англ. action, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

Рисунок 3.3 – Диаграмма деятельности

# 3.4 Диаграмма коопераций

Главная особенность диаграммы кооперации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии.

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии. На этой диаграмме не указывается время в виде отдельного измерения. Поэтому последовательность взаимодействий и параллельных потоков может быть определена с помощью порядковых номеров. Следовательно, если необходимо явно специфицировать взаимосвязи между объектами в реальном времени, лучше это делать на диаграмме последовательности.

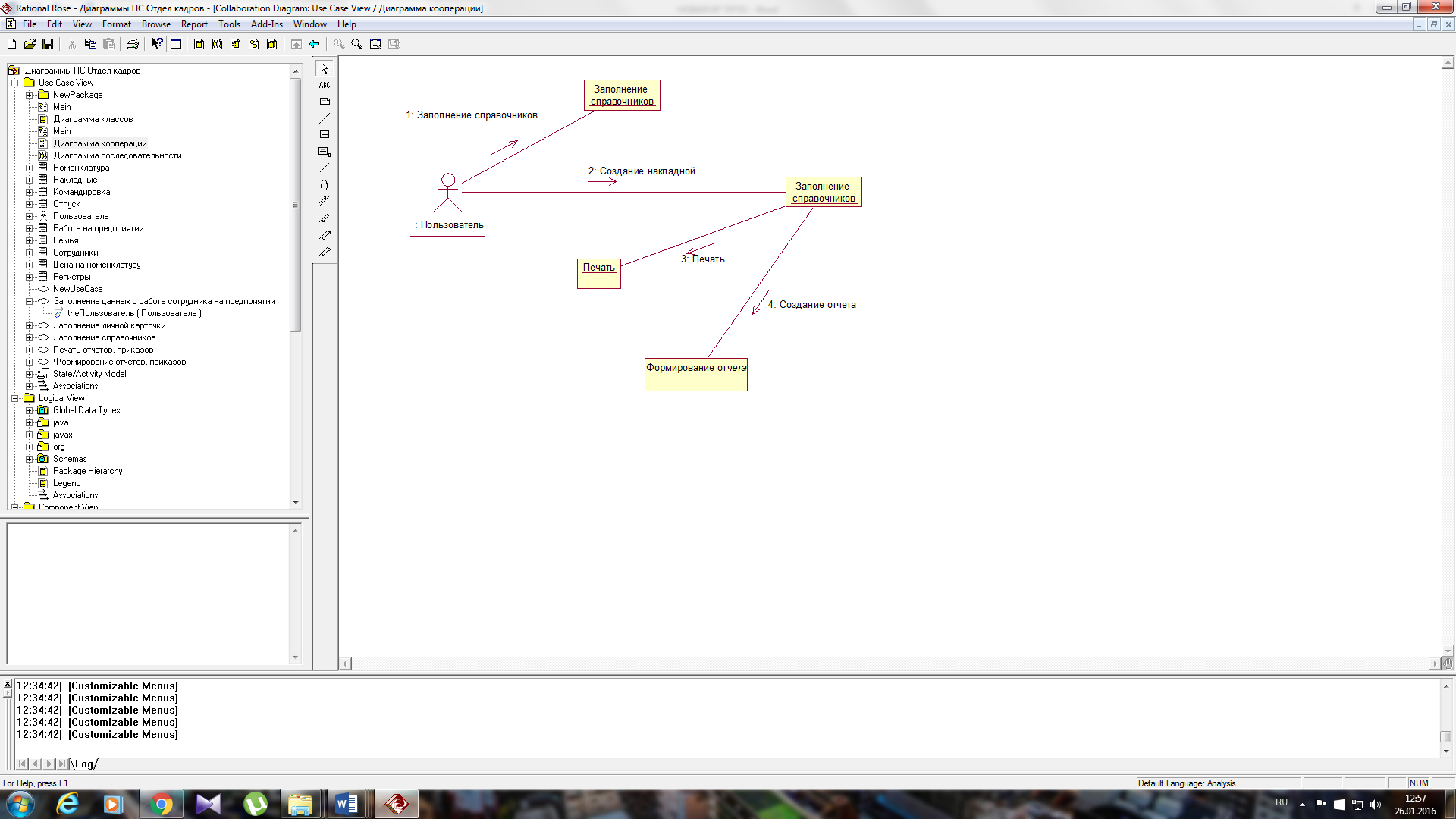


Рисунок 3.4 – Диаграмма коопераций

# 3.5 Диаграмма последовательности

Диаграмма последовательности — диаграмма, на которой показано взаимодействие объектов, упорядоченное по времени, с отражением продолжительности обработки и последовательности их проявления. Используется в языке UML.

На данной диаграмме объекты располагаются слева направо.

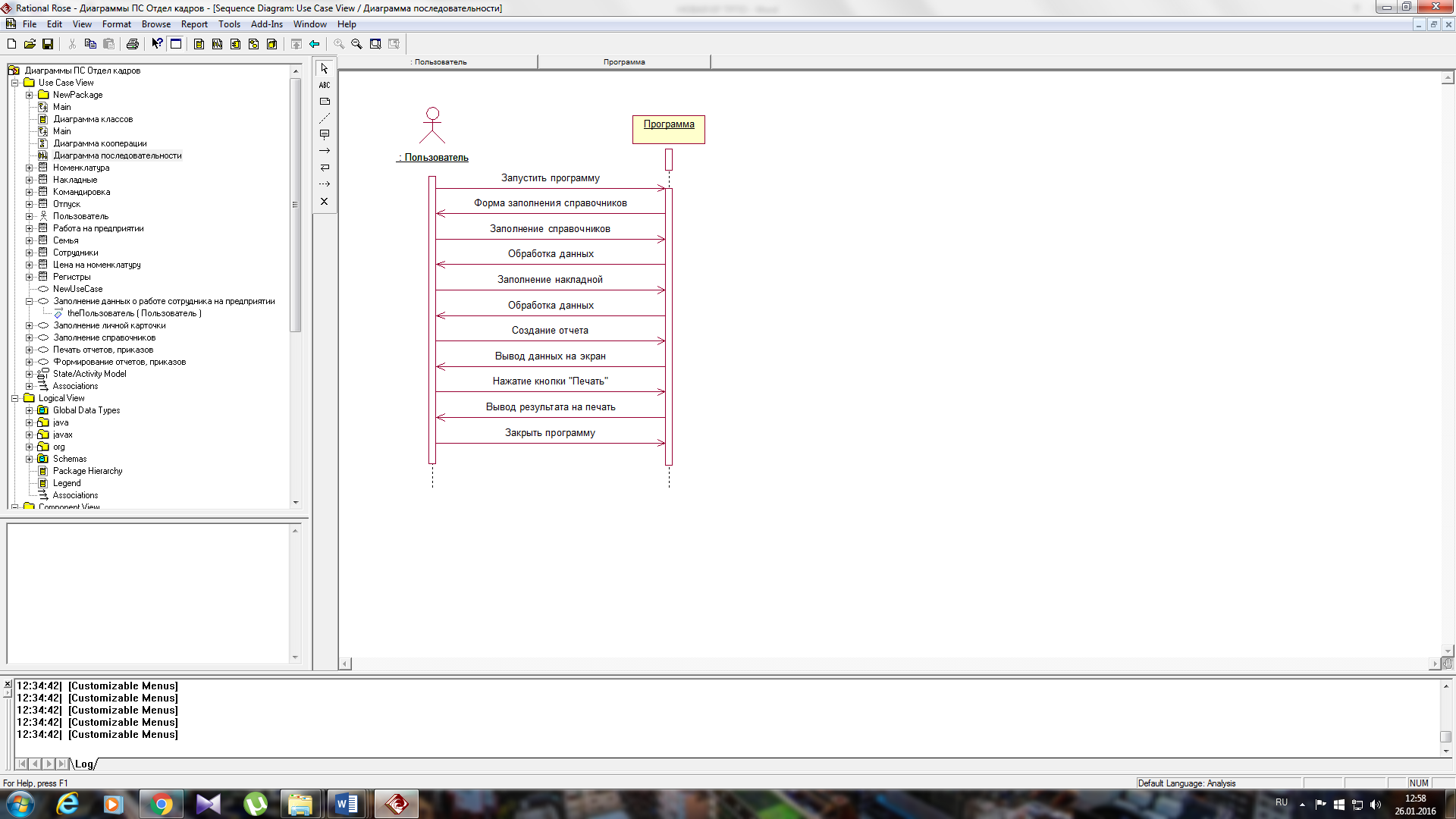


Рисунок 3.5 – Диаграмма последовательности

# 3.6 Диаграмма развёртывания

Диаграмма развёртывания в UML моделирует физическое развертывание артефактов на узлах. Например, чтобы описать веб-сайт диаграмма развертывания должна показывать, какие аппаратные компоненты («узлы») существуют (например, веб-сервер, сервер базы данных, сервер приложения), какие программные компоненты («артефакты») работают на каждом узле (например, веб-приложение, база данных), и как различные части этого комплекса соединяются друг с другом (например, JDBC, REST, RMI).

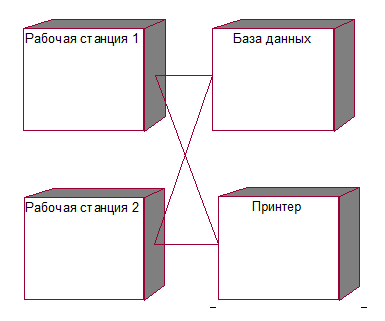


Рисунок 3.6 – Диаграмма развёртывания

# 3.7 Диаграмма состояний

Диаграмма состояний — это, по существу, диаграмма состояний из теории автоматов со стандартизированными условными обозначениями, которая может определять множество систем от компьютерных программ до бизнес-процессов.

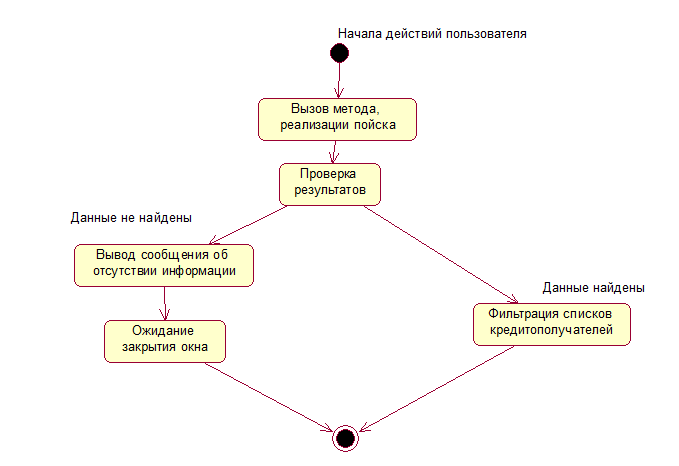


Рисунок 3.7 – Диаграмма состояний

# 3.8 Диаграмма компонентов

Диаграмма компонентов — статическая структурная диаграмма, показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т. п.

Компоненты связываются через зависимости, когда соединяется требуемый интерфейс одного компонента с имеющимся интерфейсом другого компонента. Таким образом иллюстрируются отношения клиент-источник между двумя компонентами.

Зависимость показывает, что один компонент предоставляет сервис, необходимый другому компоненту. Зависимость изображается стрелкой от интерфейса или порта клиента к импортируемому интерфейсу.

Когда диаграмма компонентов используется, чтобы показать внутреннюю структуру компонентов, предоставляемый и требуемый интерфейсы составного компонента могут делегироваться в соответствующие интерфейсы внутренних компонентов.

Делегация показывается связь внешнего контракта компонента с внутренней реализацией этого поведения внутренними компонентами.

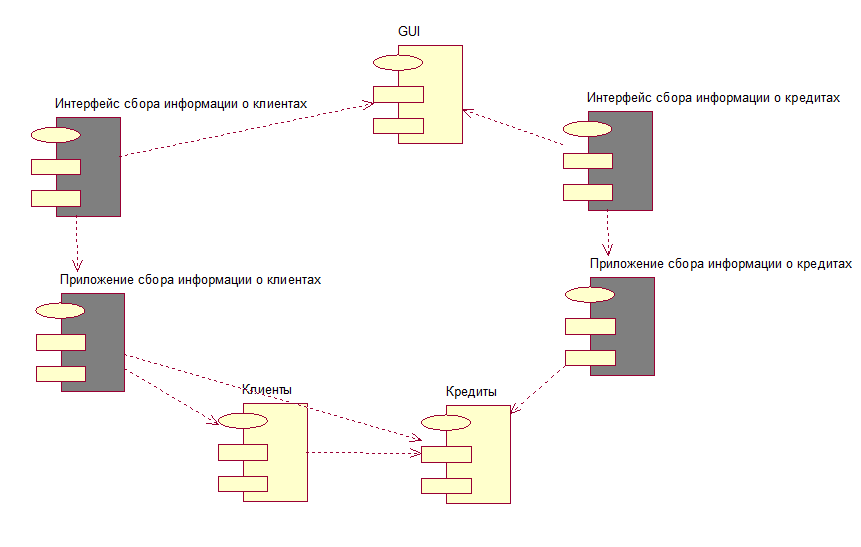


Рисунок 3.8 – Диаграмма компонентов

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Леоненков. «Самоучитель UML».

2. Б. Хичков SYBASE: Настольная книга администратора. Лори, 2000 г. 448 с.

3. Р. Мюллер. [Базы данных и UML: Проектирование](javascript:parent.descr(window,%22d160587%22)).– Лори, 2002г. 432 с.

4. Фельдман С.К. Система программирования Java без секретов: Как создать безопасное приложение с "нуля". – Новый издательский дом" , 2005 г. , 347 с.

5. Дейтел П.Дж., Дейтел Х.М. Как програмировать на Java. Книга 2. Файлы, сети, базы данных. – "Бином" · 2005 г., 672 с.

6. http://www.avacco.ru/page.asp?code=electronniy\_arhiv

7. <http://www.java.alfamoon.com/>

8. <http://www.cfin.ru/vernikov/idef/idef0.shtml>

9. http://www.cfin.ru/vernikov/idef/idef1x.shtml

# Генерация кода

Содержимое сгенерированного файла JP.java

//Source file: C:\\Jp1\\JP.java

public class Charge

{

private int cdeposit;

private int CH\_ID;

private int dateOf;

private int value;

public Logic theLogic;

/\*\*

@roseuid 587D500E006A

\*/

public Charge()

{

}

}

public class Credit

{

private int CCR\_ID;

private int closed;

private int credit;

private int customer;

private int dateOf;

private int dateOfClose;

private int value;

public Logic theLogic;

/\*\*

@roseuid 587D500D03C3

\*/

public Credit()

{

}

public class Deposit

{

private int currency;

private int D\_ID;

private int name;

private int percent;

private int summ;

public Logic theLogic;

/\*\*

@roseuid 587D500D0386

\*/

public Deposit()

{

}

}

public class Logic

{

private int filters;

private int numberOfSellected;

private int sqlOrder;

private int ado;

/\*\*

@roseuid 587D500E0021

\*/

public Logic()

{

}

}

public class Payment

{

private int credit;

private int dateOf;

private int P\_id;

private int value;

private int payment;

private int id;

public Credit theCredit;

public Logic theLogic;

/\*\*

@roseuid 587D500D0335

\*/

public Payment()

{

}

}

# Фрагмент кода программы

package logic.intf;

import util.SQLExecuteHelper;

import java.lang.reflect.Field;

import java.util.ArrayList;

public abstract class Logic implements java.io.Serializable {

protected String sqlOrder = null;

protected int numberOfSelectedFields = -1;

protected ArrayList<String> filters;

public void resetNumberOfSelectedFields() {

numberOfSelectedFields = -1;

}

public int getNumberOfSelectedFields() {

return numberOfSelectedFields;

}

public void setNumberOfSelectedFields(int numberOfSelectedFields) {

this.numberOfSelectedFields = numberOfSelectedFields;

}

public String sqlOrder() {

return sqlOrder;

}

public void setSqlOrder(String sqlOrder) {

this.sqlOrder = sqlOrder;

}

public String className() {

return this.getClass().getSimpleName();

}

private String[] getClassFieldsNames(boolean isCore) {

ArrayList<String> fieldList = new ArrayList<String>();

Field[] fields = this.getClass().getDeclaredFields();

Logic[] beans = this.getParents();

for(int fieldIndex = 0; fieldIndex < ( getNumberOfSelectedFields()!=-1 ? getNumberOfSelectedFields() : fields.length - (beans!=null ? beans.length : 0) ); fieldIndex ++) {

Field field = fields[fieldIndex];

fieldList.add(sqlTable()+"."+field.getName());

/\*

if(isCore)

fieldList.add(sqlTable()+"."+field.getDateOf());

else

fieldList.add(field.getDateOf());

\*/

}

if(beans!=null) {

for (Logic bean : beans) {

String[] beanAdditionalFields = bean.sqlRecordsNames(isCore);

String beanID = beanAdditionalFields[0];

/\*

if(isCore)

\*/

fieldList.add(sqlTable()+"."+beanID.substring(beanID.indexOf(".")+1));

/\*

else

fieldList.add(beanID);

\*/

}

}

return fieldList.toArray(new String[fieldList.size()]);

}

private String[] getClassFieldsValues() {

ArrayList<Object> fieldList = new ArrayList<Object>();

Field[] fields = this.getClass().getDeclaredFields();

Logic[] beans = this.getParents();

for(int fieldIndex = 0; fieldIndex < ( getNumberOfSelectedFields()!=-1 ? getNumberOfSelectedFields() : fields.length - (beans!=null ? beans.length : 0) ); fieldIndex ++) {

Field field = fields[fieldIndex];

field.setAccessible(true);

try {

fieldList.add(field.get(this));

} catch (IllegalAccessException e) {

e.printStackTrace();

}

}

fieldList.add( sqlOrder() );

return fieldList.toArray(new String[fieldList.size()]);

}

protected void setClassFieldsValues(String[] fieldsValues) {

Field[] fields = this.getClass().getDeclaredFields();

Logic[] beans = this.getParents();

for(int fieldIndex = 0; fieldIndex < ( getNumberOfSelectedFields()!=-1 ? getNumberOfSelectedFields() : fields.length - (beans!=null ? beans.length : 0) ); fieldIndex ++) {

Field field = fields[fieldIndex];

field.setAccessible(true);

try {

field.set(this, fieldsValues[fieldIndex]);

} catch (IllegalAccessException e) {

e.printStackTrace();

}

}

this.setSqlOrder( fieldsValues[fieldsValues.length-1] );

}

// --------------------

public String[] sqlRecordsNames(boolean isCore) {

return getClassFieldsNames(isCore);

}

public String[] sqlRecordsValues() {

return getClassFieldsValues();

}

public Logic instanceFromRecordsValues(String[] params) {

return instanceFromRecordsValues(params, -1);

}

public abstract Logic instanceFromRecordsValues(String[] params, int numberOfSelectedFields);

// --------------------

public String sqlIDName() {

String sqlIDName = getClassFieldsNames(false)[0];

return sqlIDName.substring(sqlIDName.indexOf(".")+1);

}

public abstract String[] getCaptions();

public abstract String getLogicName();

public abstract int[] getWidthes();

public abstract String ID();

public Logic[] getParents() {

return null;

}

public void initBean(Logic logic, int beanIndex) {

}

public abstract LogicFieldTypes[] getTypes();

public String sqlParentIDName() {

return null;

}

public String sqlTable() {

return className();

}

public String sqlFrom(SQLExecuteHelper sqlExecuteHelper) {

StringBuffer Period = new StringBuffer();

StringBuffer where = new StringBuffer();

String className = className();

Period.append(className);

if(getParents() != null) {

Period.append(",");

for(Logic bean : getParents()) {

Period.append(bean.sqlTable()).append(",");

where.append(" and ");

where.append(sqlTable()).append(".").append(bean.sqlIDName())

.append("=")

.append(bean.sqlTable()).append(".").append(bean.sqlIDName());

if(isElementFilled(bean)) {

where.append(" and ")

.append(sqlTable()).append(".").append(bean.sqlIDName())

.append(" in ")

.append(" ( select ")

.append(bean.sqlIDName()).append(" from ")

.append(bean.sqlFrom(sqlExecuteHelper));

String[] recordNames = bean.sqlRecordsNames(true);

String[] recordValues = bean.sqlRecordsValues();

for(int recordIndex = 0; recordIndex < recordNames.length-1; recordIndex++) {

if(recordValues[recordIndex]!=null && !recordValues[recordIndex].equals("")) {

where.append(" and ");

where.append(recordNames[recordIndex]);

if(recordNames[recordIndex].equals(bean.sqlTable()+"."+bean.sqlIDName())) {

where.append(" = ? ");

sqlExecuteHelper.addParam(recordValues[recordIndex]);

} else {

where.append(" like ? ");

sqlExecuteHelper.addParam("%"+recordValues[recordIndex]+"%");

}

}

}

where.append(" ) ");

} else {

}

/\*

where.append("=").

append(bean.sqlTable()).append(".").append(bean.sqlIDName());

\*/

}

Period.deleteCharAt(Period.length()-1);

}

Period.append(" where 1=1 ");

Period.append(where);

return Period.toString();

}

private boolean isElementFilled(Logic element) {

boolean isFilled = false;

String[] sqlRecordsValues = element.sqlRecordsValues();

for(int recordIndex = 0; recordIndex < sqlRecordsValues.length-1; recordIndex++) {

String recordValue = sqlRecordsValues[recordIndex];

if(recordValue!=null && recordValue.trim().length()!=0) {

isFilled = true;

break;

}

}

return isFilled;

}

public class LogicParentIndexException extends RuntimeException{

public String toString() {

return "Wrong parent index";

}

}

public String getFilterPresentation() {

return sqlRecordsValues()[1];

}

}