Министерство образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

**Институт Информационных технологий**

Специальность «Информационные системы и технологии (в экономике)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ**

По курсу: «Программирование сетевых приложений»

Тема: «Разработка автоматизированной системы пункта проката видеофильмов»

Студент-заочник 3 курса

Группы № 382372

Григорук Александр Владимирович

Адрес: г. Минск,

ул. Почтовая, д. 3, кв. 64

E-mail: thanksforsource@gmail.com

Тел. +375 29 694 41 53

Проверил: Д.А.Сторожев

Минск, 2016

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc443690561)

[1. Описание предметной области и определение требований к системе 4](#_Toc443690562)

[1.1 Описание предметной области 4](#_Toc443690563)

[1.2 Определение требований к системе 5](#_Toc443690564)

[2. Постановка задачи и рассмотрение методов решения 6](#_Toc443690565)

[2.1 Постановка задачи 6](#_Toc443690566)

[2.2 Обзор способов решения задачи 6](#_Toc443690567)

[3. Модели представления системы и их описание. Архитектура клиент-сервер. 7](#_Toc443690568)

[3.1 Модели представления системы и их описание 7](#_Toc443690569)

[3.2 Архитектура клиент-сервер. 8](#_Toc443690570)

[4. Информационная модель системы и ее описание 9](#_Toc443690571)

[4.1 Описание информационной модели 9](#_Toc443690572)

[5. Описание решений по использованию технических и программных средств, не включенных в требования 11](#_Toc443690573)

[6. Описание бизнес-логики серверной части системы и ее взаимодействие………………………………………………………………….13](#_Toc443690574)

[7. Руководство пользователя 14](#_Toc443690575)

[ЗАКЛЮЧЕНИЕ 17](#_Toc443690576)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 18](#_Toc443690577)

[ПРИЛОЖЕНИЕ А 19](#_Toc443690578)

[Функциональная модель системы 19](#_Toc443690579)

[ПРИЛОЖЕНИЕ Б 21](#_Toc443690580)

[Диаграмма вариантов использования 21](#_Toc443690581)

[ПРИЛОЖЕНИЕ В 22](#_Toc443690582)

[SQL –скрипт структуры целевой базы данных 22](#_Toc443690583)

[ПРИЛОЖЕНИЕ Г 26](#_Toc443690584)

[Диаграмма состояний 26](#_Toc443690585)

[ПРИЛОЖЕНИЕ Д 27](#_Toc443690586)

[Диаграмма последовательностей 27](#_Toc443690587)

[ПРИЛОЖЕНИЕ Е 28](#_Toc443690588)

[Листинг основных функций программы 28](#_Toc443690589)

# ВВЕДЕНИЕ

В современном мире бизнес уже не способен обходится без компьютеров. Отчеты, сведения о продажах, слежение за рынком сбыта – все это уже давно перешло в эру компьютерной обработки. Многие фирмы желают обзавестись своими информационными продуктами, которые помогут выполнять необходимые для бизнеса функции быстрее и без участия человека. Средства связи позволили расширить рынок вещей, интернет – приблизил потенциального покупателя к продавцу. Именно поэтому сейчас создание программ для автоматизации определенных бизнес процессов и сайты для продвижения и рекламы стали очень востребованы.

Целью данного курсового проекта является одна из систем автоматизации. А именно автоматизации пункта проката видеоматериалов. Для выполнения данной работы мы определим задачи, которые необходимо решить:

- выполнить анализ предметной области;

- построить UML диаграммы, которые будут описывать реализацию;

- построить базу данных для хранения и передачи данных;

- создать программное обеспечение, реализующее данную задачу c использованием БД в качестве хранилища данных;

- использовать архитектуру клиент-сервера для реализации задачи;

В ходе выполнения курсового проекта будут получены знания и опыт разработки приложений на языке Java, использования IDE Eclipse, реализации хранилища данных в СУБД MySQL, а также использования шаблонов проектирования.

1. Описание предметной области и определение требований к системе
   1. Описание предметной области

Предметная область данного проекта – это осуществление предпринимательской деятельности по аренде видеоматериалов физическим лицам. В целом весь процесс можно описать в шагах, которые нужно предпринять потенциальному клиенту, чтобы получить услугу (товар).

Опишем процесс:

1. Клиент приходит в пункт проката и выбирает фильм, который он желает взять на прокат;
2. Персонал регистрирует клиента в своей системе, используя уникальные данные от пользователя ( Ф.И.О клиента, телефон и номер паспорта);
3. Далее клиент указывает, на какой период он желает приобрести аренду. Исчисляется в днях от момента предоставления услуги;
4. Персонал, используя программное обеспечение, высчитывает сумму к оплате и производит прием платежа;
5. Клиент оплачивает заказ и получает услугу(товар) на оговоренный срок;
6. После истечения срока клиент обязан вернуть товар в пункт аренды, где возврат товара зарегистрируют в системе.

В разрабатываемом программном продукте будет предусмотрен функционал, позволяющий реализовать каждый шаг данного процесса.

* 1. Определение требований к системе

Требования к разрабатываемому программному обеспечению выглядят следующим образом:

1. Простой пользовательский интерфейс, однако реализующий весь необходимый доступ ко всем функциям программы;
2. Для реализации хранения данных будет выбрана СУБД MySQL ;
3. Для возможности развертывания на разных машинах, база данных должна генерироваться из backup файла или же через sql скрипт;
4. Использовать архитектуру клиент-сервера. Все, что реализовано на сервере, доступно на клиенте через специальный интерфейс
5. Для реализации связи между клиентом и сервером использовать технологию RMI
6. Постановка задачи и рассмотрение методов решения
   1. Постановка задачи

Целью данной курсовой работы является разработка автоматизированной системы пункта проката видеоматериалов. Она будет предназначаться для облегчения и работы сотрудников проката. Используя данную программу, сотрудник сможет быстро составить заказ, зарегистрировать покупателя в системе, а также проверить в наличии ли определенный фильм.

* 1. Обзор способов решения задачи

Для решения данной задачи необходимо использовать следующие методы:

1. Анализ предметной области с построением диаграммы IDEF0.
2. Создание сервера приложения. Его задачей будет взаимодействие с БД и формирование данных для обмена с клиентом.
3. Создание клиентской части: GUI приложение.
4. Создание архитектуры базы данных. Задача БД – хранение данных.
5. Использование прослойки-драйвера для связи выбранной СУБД с создаваемым приложением. Т.к. у нас приложение создается посредством языка JAVA, то в качестве драйвера будет использоваться JDBC драйвер.
6. Модели представления системы и их описание. Архитектура клиент-сервер.
   1. Модели представления системы и их описание

Посредством диаграмм можно описать программу с разных сторон. Опишем несколько из них и начнем с диаграммы вариантов использования.

Программой пользуется оператор (продавец), который с её помощью способен проводить процесс регистрации заказа и выдачи товара заказчику. Он имеет доступ ко всем функциональным частям программы – способен просмотреть уже существующие заказы, просмотреть список фильмов в наличии, произвести поиск по нему. Также, оператор способен зарегистрировать новый заказ на выдачу видеоматериала или же зафиксировать возврат заказа по истечению аренды. Диаграмма вариантов использования (прецедентов) описана в приложении Б.

Диаграмма состояний позволяет указать основные моменты работы программы ( состояния) за весь жизненный цикл. В данном случае мы будем описывать цикл заказа начиная от его создания, заполнения и подтверждения, а также его закрытия ( товар возвращен). Данная диаграмма описана в приложении Г.

Диаграмма последовательности действий описывает порядок выполнения операций, которые необходимо произвести для использования определенного функционала программы. В данной работе будут описаны действия оператора для взаимодействия с программой. Диаграмма описана в приложении.

* 1. Архитектура клиент-сервер.

Архитектура данного проекта была выбрана исходя из задания по реализации. А именно требованием разделить функционал программы между составными частями – клиентом и сервером. Поставщик услуг или сервер отвечает за обработку информации и формировании моделей, пригодных и адаптированных для конечного клиента. Заказчик услуг или клиент представляет собой оболочку (обычно с графическим пользовательским интерфейсом GUI ), которая общается с сервером посредством сетевых протоколов через сеть и предоставляет возможность пользователю системы взаимодействовать с ней.

Основным преимуществом данной архитектуры являются несколько моментов. Одни из них является возможность избежать дублирования кода из серверной части в клиентской. Данный момент поддерживает один из принципов разработки DRY, что означает «don’t repeat yourself» или «не повторяйся». Другим важным аспектом данной архитектуры является распределение нагрузки. Так как сервер и клиент разделены, то обработка данных и их построение в большинстве случаев переносится на сервер. Сами серверные части располагаются на компьютерах более специализированных на обработке данных. И это позволяет уменьшить нагрузку на пользовательские компьютеры. Также использование клиент-сервера обеспечивает необходимый уровень защиты данных от несанкционированного доступа т.к. компьютеры сервером защищены гораздо сильнее, нежели клиентские.

К недостаткам можно отнести зависимость от сети, высокую стоимость серверного оборудования и поддержки системы.

1. Информационная модель системы и ее описание
   1. Описание информационной модели

Опишем информационную модель, которая необходима для построения приложения. При проектировании и разработки системы были выделены следующие сущности:

1. Films – сущность, описывающая собой модель товара (видеоматериалов). Имеет следующие атрибуты: FilmId – уникальный идентификатор, Name – наименование, Type – категория (жанр), AvailableCount – количество товара в наличии, Rating – оценка товара, RentCost – цена аренды за единицу времени (календарный день), Year – год издания.
2. Customer – сущность клиента или заказчика. Имеет следующие атрибуты: CustomerId - уникальный идентификатор, FirstName – имя клиента, LastName – фамилия клиента, PassportNumber – пасспортный номер, Discount – скидка на аренду, PhoneNumber – номер телефона.
3. Order – сущность заказа. Имеет следующие атрибуты:OrderId –уникальный идентификатор (первичный ключ PK), СustomerId – вторичный ключ (FK) ссылающийся на сущность Customer,FilmId – вторичный ключ (FK) ссылающийся на сущность Films, Created – дата создания, RentExpires – дата окончания аренды (дата возврата),Returned – состояния заказа (возвращен или нет).
4. Employee – сущность персонала. Имеет следующие атрибуты:EmployeeId – уникальный идентификатор, CashierPassword – пароль, для доступа к функционалу программы,Hired – дата найма, ContractExpires – дата окончания контракта, FirstName- имя работника, LastName – фамилия, Login – логин, необходимый для доступа к функционалу программы.

Описанная выше модель представлена на рисунке 4.1.

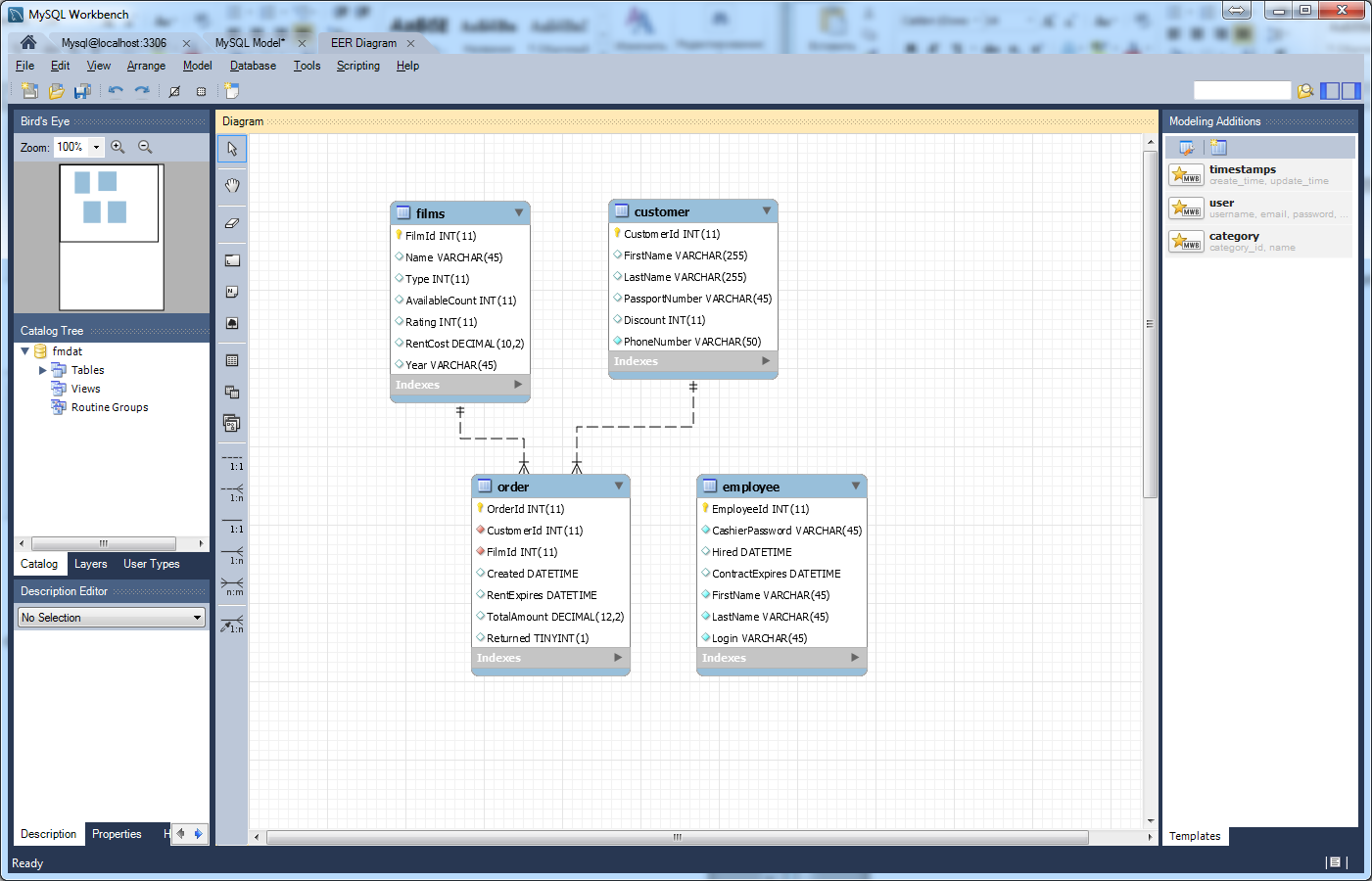


Рисунок 4.1 – Информационная модель системы.

1. Описание решений по использованию технических и программных средств, не включенных в требования

Одним из нестандартных решений является взаимодействие сервера и клиента. Данный функционал реализован с помощью протокола удаленного вызова методов RMI.

В контексте RMI есть определенные объекты, которые можно разделить по их принадлежности на клиентский объект и серверный. Задача серверного объекта – предоставить удаленный метод для клиентского объекта. Компьютеры выступают в роли сервера и клиента только для конкретного вызова в определенный момент времени. Они способны поменяться ролями если это необходимо. То есть сервер предыдущего вызова может сам стать клиентом при обращении к объекту другого компьютера.

Если рассмотреть поближе сам процесс вызова удаленного метода, то можно заметить некоторые особенности реализации. На самом деле во время компиляции на стороне клиента уже существует объект называемый заглушкой (Stub), которая является имитацией такого же объекта на серверной стороне. Т.е. есть описание вызываемого метода, но он инкапсулирован в заглушке на клиентском компьютере.

Заглушка упаковывает параметры удаленного метода в блок байтов. Каждый параметр кодируется с помощью алгоритма, обеспечивающего независимость от аппаратуры. Например, числа всегда передаются в порядке, при котором сначала передается старший байт (big-endian). При этом объекты подвергаются сериализации. Процесс кодирования параметров называется развертыванием параметров (parameter marshaling). Основная цель развертывания параметров — преобразование их в формат, пригодный для передачи параметров от одной виртуальной машины к другой.

В данной работе RMI был применен из-за простоты реализации т.к. необходимые методы были вынесены в интерфейс, который был скопирован на клиентскую часть приложения. И для реализации работы RMI необходимо было реализовать интерфейс Remote на стороне сервера. Далее выбрать порт и «привязать» (bind) именованный вызов, с помощью которого будет доступен вызов удаленного метода (например localhost/getRepo) .

1. Описание бизнес-логики серверной части системы и ее взаимодействие

Для обеспечения взаимодействия с БД используется класс DataBaseProvider. В основе его работы лежит использование JDBC драйвера для соединения с БД. Из-за особенности его реализации, объект данного класса будет создан лишь единожды при старте приложения, и дальше будет использоваться только он. Это реализация шаблона проектирования Singleton или одиночка. Плюсом является то, что мы не создаем для каждого вызова бд новый объект класса, а выдаем доступ к уже сформированному соединению через метод.

Сущности БД на серверно стороне представлен классами Enities, которые имеют схожую структуру с самими таблицами в БД. Они реализуют интерфейс Serializable для того, чтобы было возможность упаковать данные при передаче их через сеть на сторону клиента. Для взаимодействия с БД CRUD операции (Create,Read,Update,Delete) описаны в репозиториях. Каждый из репозиториев имеет собственный интерфейс для того, чтобы реализовать DI. Расшифровывается как Dependency Injection (внедрение зависимостей). Его особенность заключается в том, что мы передаем управление созданием объекта какому то общему механизму. В нашем случае это сервис ConnectService, который инциализирует все репозитории в одном месте при его вызове. В данном случае был использован механизм под названием Dependency Injection Container. Это означает что мы передали управление в конструктор общего сервиса.

Все необходимые вызовы для клиента описаны в сервисе ConnectService. И в данной работе используется интерфейс этого сервиса (IConnectService), как связующее звено между сервером и клиентом.

1. Руководство пользователя

Перед началом использования клиентской части приложения, необходимо запустить серверную. Она представляет собой GUI приложение с окном, в котором описываются текущие процессы на сервере (рис. 7.1).

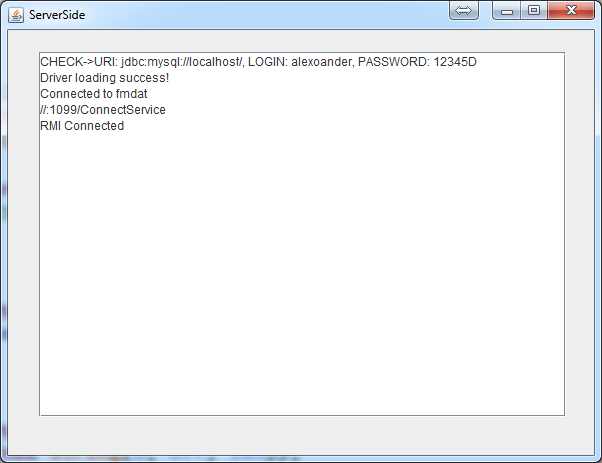


Рисунок 7.1 – Внешний вид серверной части программы

После того, как запущена серверная часть и в статусе написано «RMI Connected» можно запускать клиентскую часть программы.

При запуске клиент сразу же получит приглашение ввести логин и пароль для доступа к функционалу программы (рис.7.2).

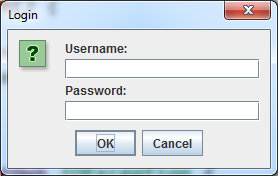


Рисунок 7.2 – Форма логина

После ввода верных данных ( они регистрируются в БД заранее), оператор получает доступ к самой программе (рис.7.3).

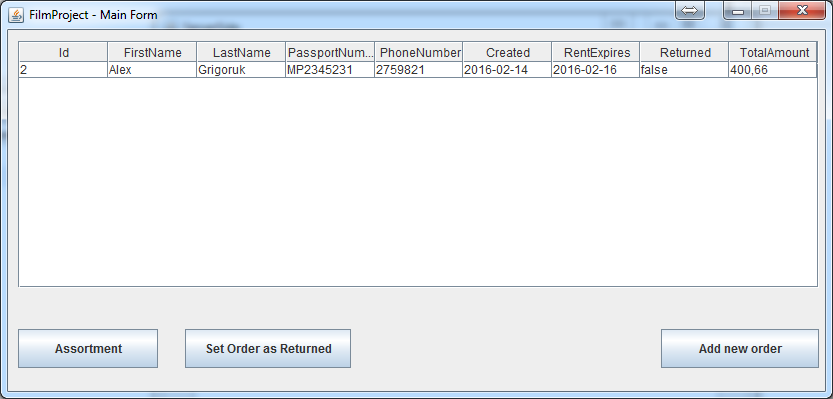


Рисунок 7.3 – Внешний вид клиентской части приложения

Оператор может просмотреть список фильмов, доступных к реализации, нажав на кнопку Assortment (рис. 7.4). На главной форме можно увидеть список всех заказов, которые уже были созданы. Главной особенностью является указание даты заказа и даты истечения аренды. Если клиент вернул товар, оператор выделяет нужный заказ и нажимает на кнопку «Set Order as Returned». Это зафиксирует факт возврата товара и заказ будет завершен.

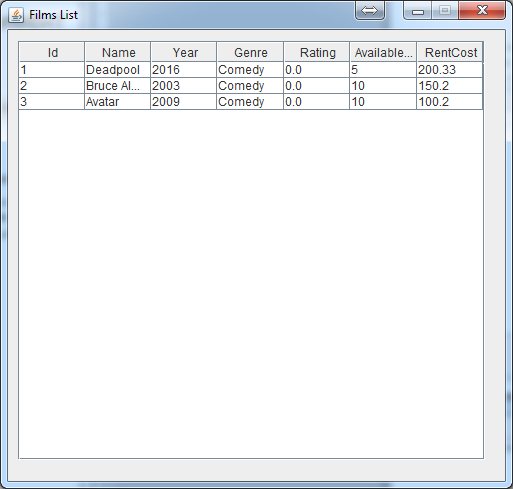


Рисунок 7.4 – Внешний вид списка фильмов

При нажатии на кнопку «Add new order» появится форма для создания нового заказа (рис. 7.5). С её помощью оператор сможет зарегистрировать клиента в системе и создать заказ. При выборе нужного фильма и указании срока аренды ( в днях), программа автоматически посчитает необходимую сумму.

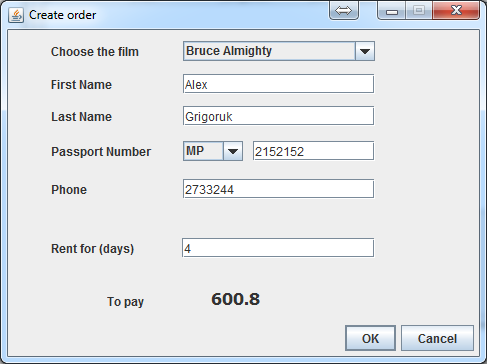


Рисунок 7.5 – Внешний вид формы создания заказа

# ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта была создана система, которая удовлетворяет поставленным задачам. Приложение позволяет быстро и эффективно создавать и отслеживать заказы, а также не требует специального обучения для использования. В процессе создания программы были изучены многие детали реализации программ на языке Java в IDE Eclipse, использование Swing для построения пользовательского интерфейса, особенности разработки клиент-серверных систем.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1]Хорстман, М. Java-2. Библиотека профессионала. Том 1 / Хорстман**.–**М.:Вильямс, 2003. – 850с.

[2]Хорстман, М. Java-2. Библиотека профессионала. Том 2 / Хорстман**.–**М.:Вильямс, 2003. – 630с.

[3]Дейтел, - М. Технологии программирования на Java 2. Том 1-3 / Х.М.Дейтел, П.Дж.Дейтел. – М.: Бином, 2003 – 1560с.

[4]Буч, - М. UMLРуководство пользователя / Г.Буч. – М.: Вильямс, 2005. – 260с.

# ПРИЛОЖЕНИЕ А

# Функциональная модель системы

(обязательное)

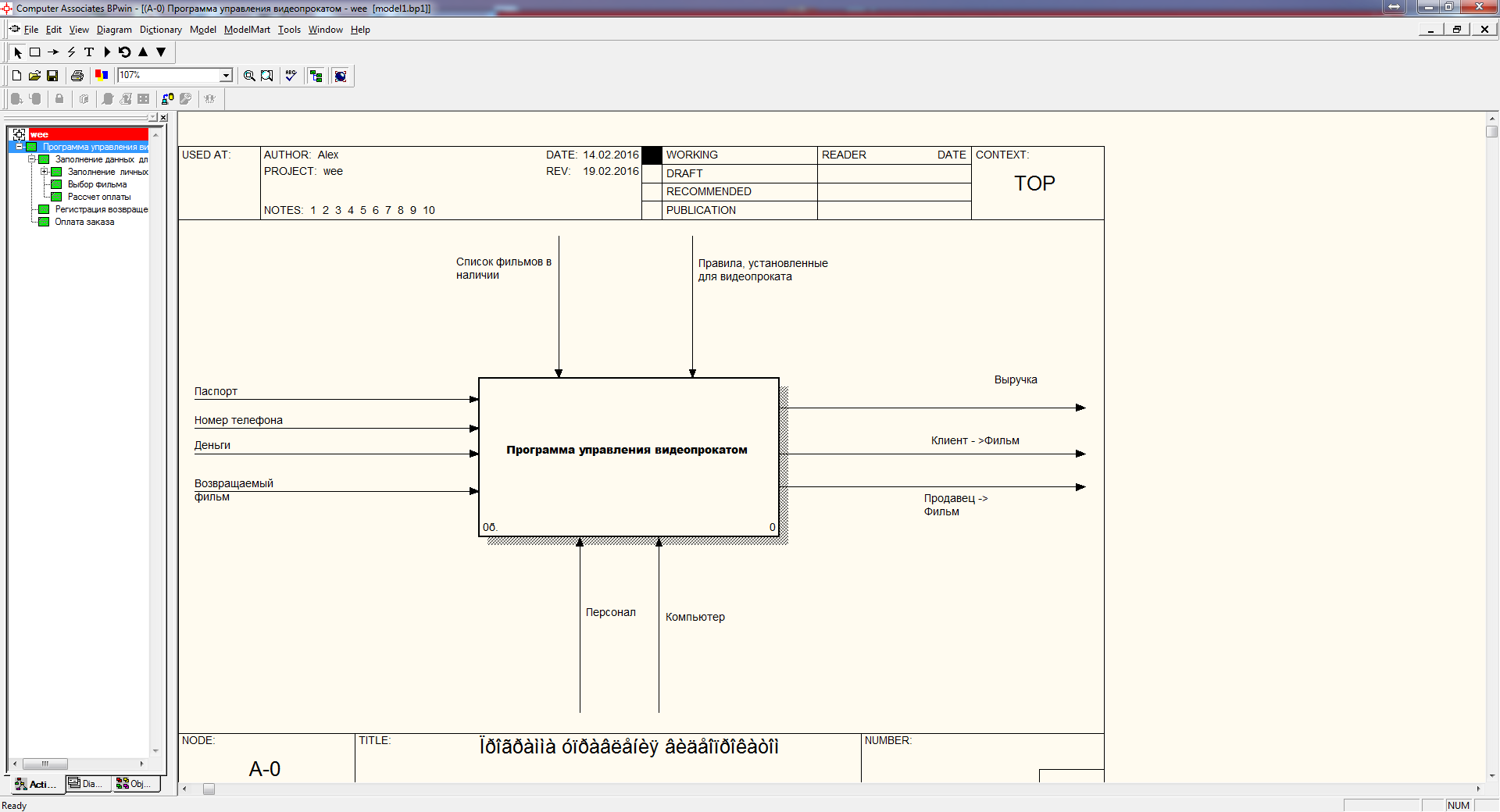


Рисунок А1 – Нулевой уровень модели IDEF0 (A-0)

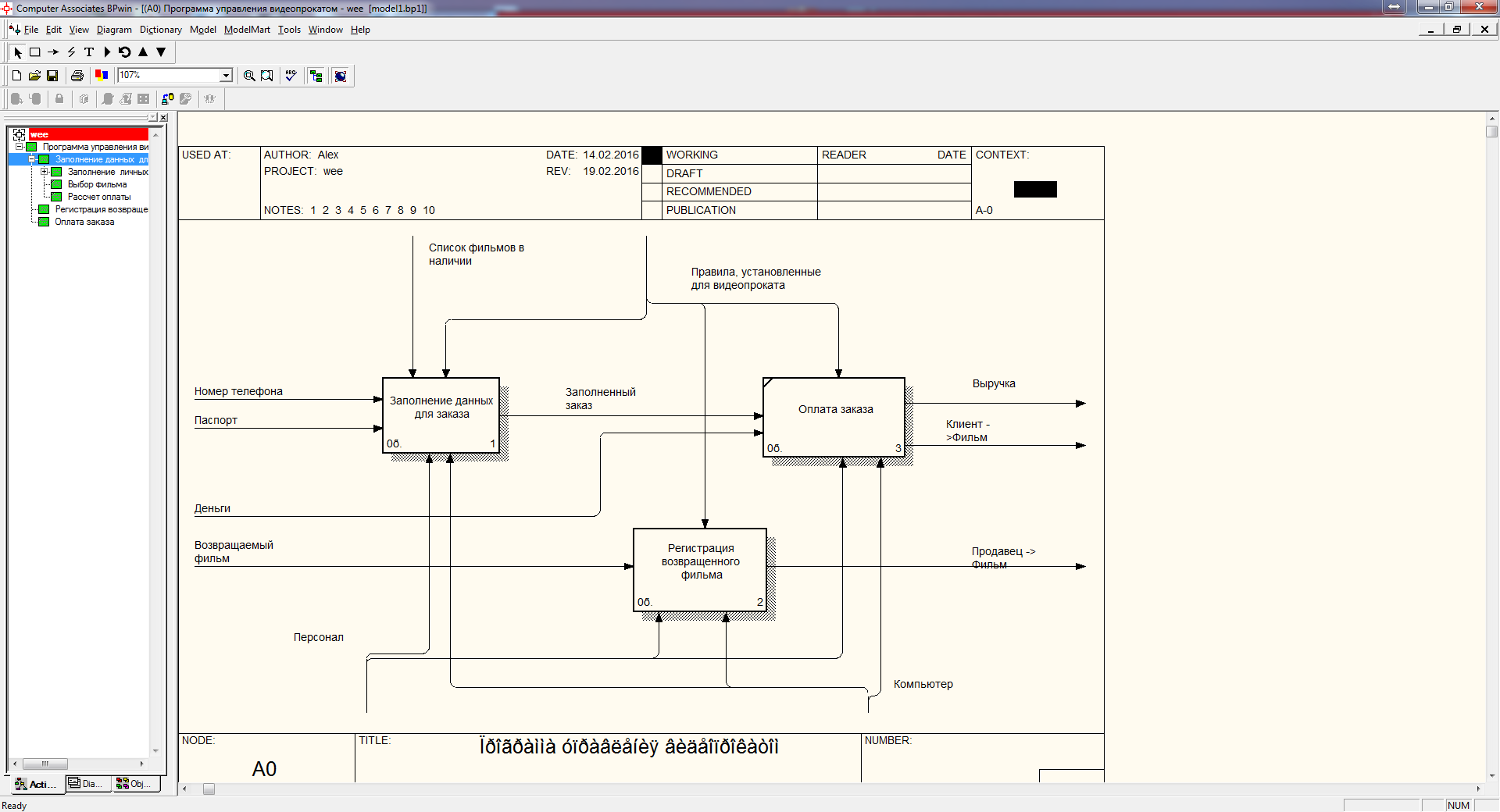


Рисунок A2 –Первый уровень декомпозиции IDEF0 (A0)

Продолжение приложения А

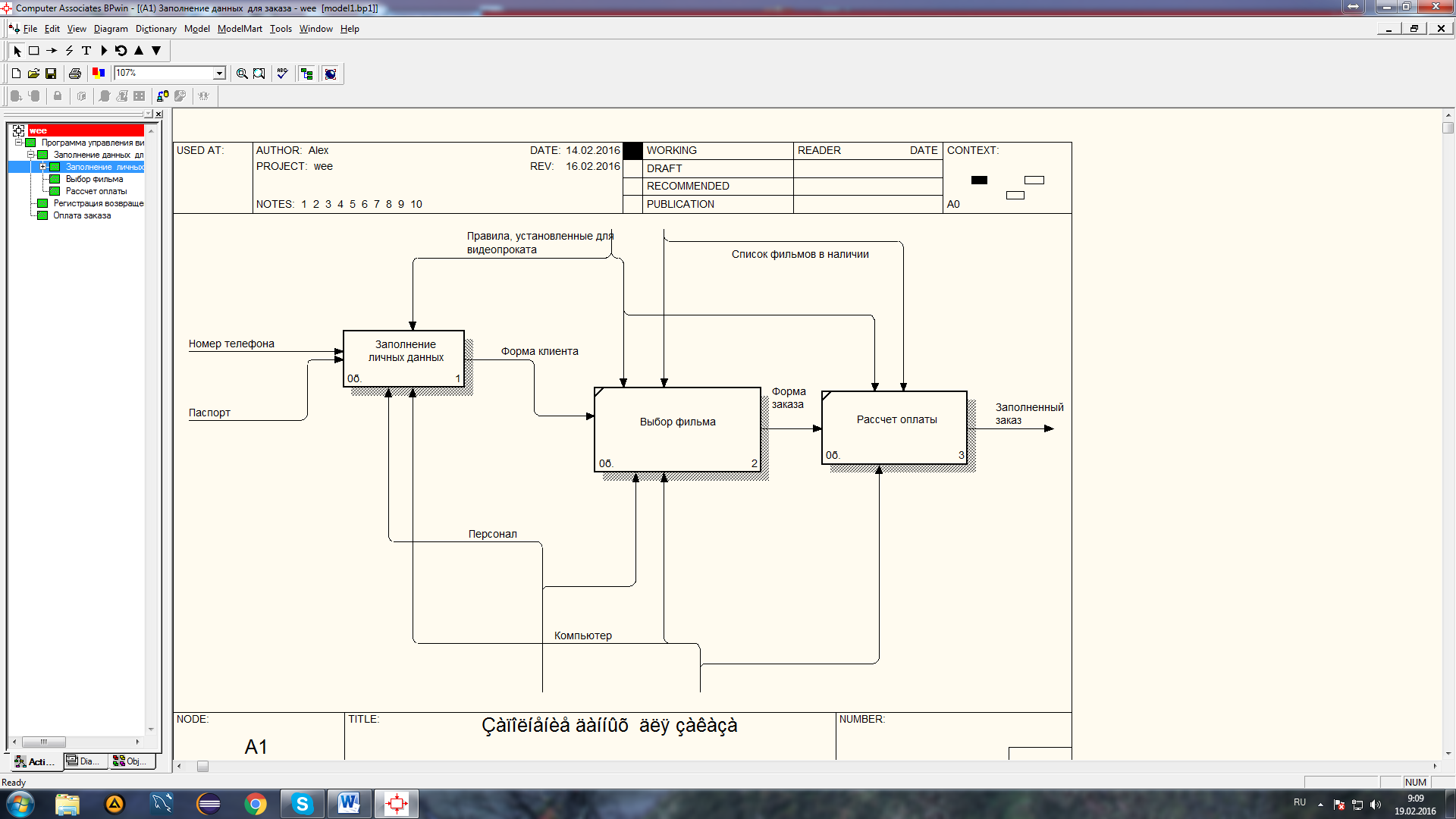


Рисунок А3 – Второй уровень декомпозиции (A1)

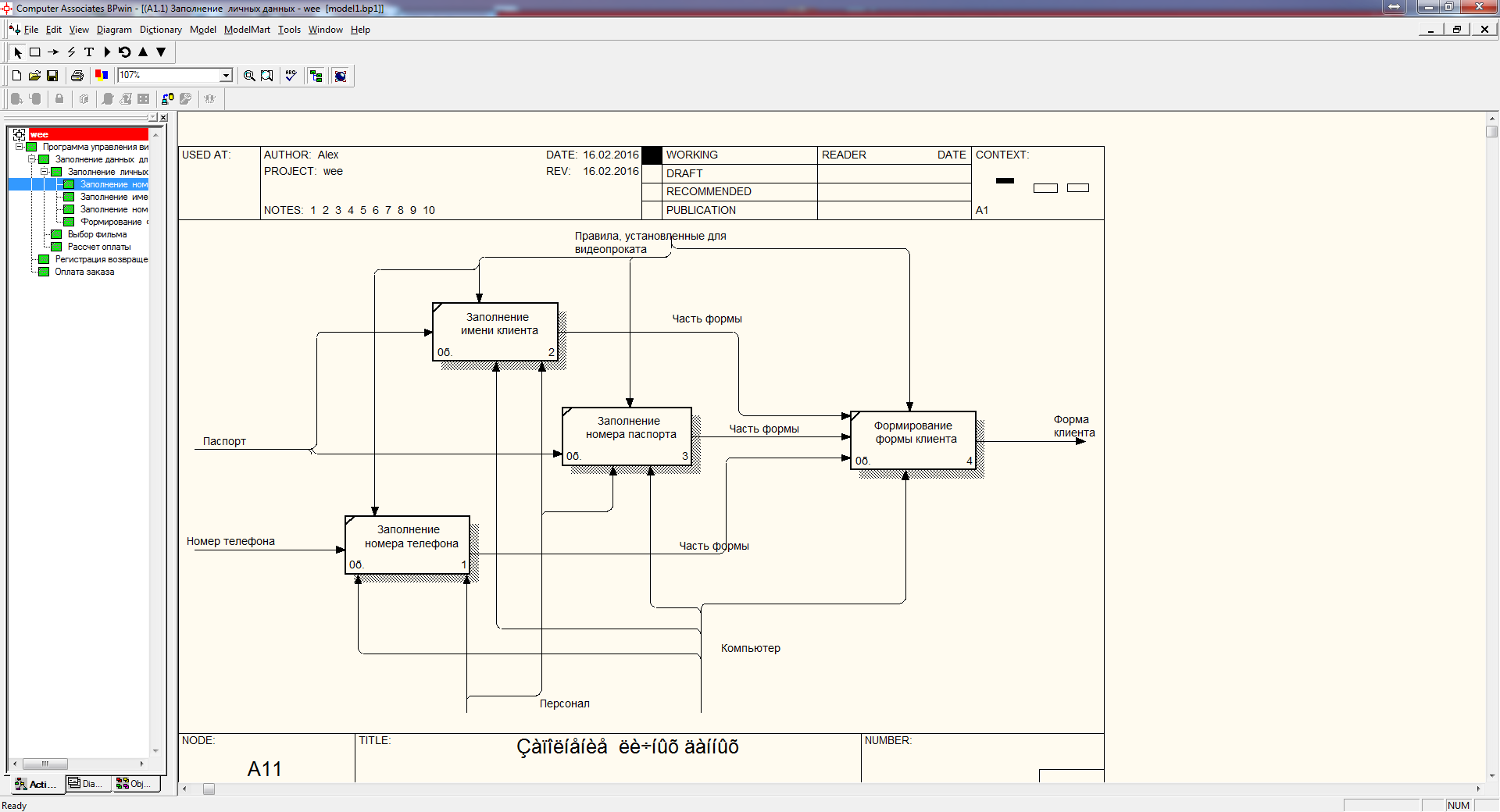


Рисунок А4 – Третий уровень декомпозиции (A11)

# ПРИЛОЖЕНИЕ Б

# Диаграмма вариантов использования

(обязательное)

User

Оформить заказ

Закрыть заказ

Просмотреть список фильмов

Рисунок Б.1 – Диаграмма прецедентов

# ПРИЛОЖЕНИЕ В

# SQL –скрипт структуры целевой базы данных

(обязательное)

DROP TABLE IF EXISTS `customer`;

CREATE TABLE `customer` (

`CustomerId` int(11) NOT NULL AUTO\_INCREMENT,

`FirstName` varchar(255) DEFAULT NULL,

`LastName` varchar(255) DEFAULT NULL,

`PassportNumber` varchar(45) DEFAULT NULL,

`Discount` int(11) DEFAULT NULL,

`PhoneNumber` varchar(50) NOT NULL DEFAULT '0-000-000',

PRIMARY KEY (`CustomerId`)

) ENGINE=InnoDB AUTO\_INCREMENT=15 DEFAULT CHARSET=cp1250;

LOCK TABLES `customer` WRITE;

INSERT INTO `customer` VALUES (14,'Alex','Grigoruk','MP2345231',0,'2759821');

/\*!40000 ALTER TABLE `customer` ENABLE KEYS \*/;

UNLOCK TABLES;

CREATE TABLE `employee` (

`EmployeeId` int(11) NOT NULL AUTO\_INCREMENT,

`CashierPassword` varchar(45) NOT NULL,

`Hired` datetime DEFAULT NULL,

`ContractExpires` datetime DEFAULT NULL,

`FirstName` varchar(45) NOT NULL,

`LastName` varchar(45) NOT NULL,

`Login` varchar(45) NOT NULL,

PRIMARY KEY (`EmployeeId`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8;

LOCK TABLES `employee` WRITE;

/\*!40000 ALTER TABLE `employee` DISABLE KEYS \*/;

INSERT INTO `employee` VALUES (1,'2233','2016-02-14 00:00:00','2017-02-14 00:00:00','Squall','Lionheart','lionheart');

/\*!40000 ALTER TABLE `employee` ENABLE KEYS \*/;

UNLOCK TABLES;

DROP TABLE IF EXISTS `films`;

CREATE TABLE `films` (

`FilmId` int(11) NOT NULL AUTO\_INCREMENT,

`Name` varchar(45) DEFAULT NULL,

`Type` int(11) DEFAULT NULL,

`AvailableCount` int(11) DEFAULT NULL,

`Rating` int(11) DEFAULT NULL,

`RentCost` decimal(10,2) DEFAULT NULL,

`Year` varchar(45) DEFAULT NULL,

PRIMARY KEY (`FilmId`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

LOCK TABLES `films` WRITE;

INSERT INTO `films` VALUES (1,'Deadpool',1,5,5,200.33,'2016'),(2,'Bruce Almighty',1,10,5,150.20,'2003'),(3,'Avatar',1,10,5,100.20,'2009');

/\*!40000 ALTER TABLE `films` ENABLE KEYS \*/;

UNLOCK TABLES;

DROP TABLE IF EXISTS `order`;

CREATE TABLE `order` (

`OrderId` int(11) NOT NULL AUTO\_INCREMENT,

`CustomerId` int(11) NOT NULL,

`FilmId` int(11) NOT NULL,

`Created` datetime DEFAULT NULL,

`RentExpires` datetime DEFAULT NULL,

`TotalAmount` decimal(12,2) DEFAULT '0.00',

`Returned` tinyint(1) DEFAULT '0',

PRIMARY KEY (`OrderId`),

KEY `FK\_order\_customer\_idx` (`CustomerId`),

KEY `FK\_order\_film\_idx` (`FilmId`),

CONSTRAINT `FK\_order\_customer` FOREIGN KEY (`CustomerId`) REFERENCES `customer` (`CustomerId`) ON UPDATE CASCADE,

CONSTRAINT `FK\_order\_film` FOREIGN KEY (`FilmId`) REFERENCES `films` (`FilmId`) ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=cp1251;

# ПРИЛОЖЕНИЕ Г

# Диаграмма состояний

(обязательное)

Запуск программы

Авторизация

Пароль верный

Пароль неверный

Главное меню

Закрыть окно

Кнопка «Close Order»

Кнопка «Assortment»

Закрытие существующего заказа

Просмотр списка фильмов

Кнопка «Create Order»

Закрыть окно

Кнопка «Сancel»

Форма нового заказа

Кнопка «Ок»

Добавление нового заказа

Рисунок Г.1 – Диаграмма состояний

# ПРИЛОЖЕНИЕ Д

# Диаграмма последовательностей

(обязательное)

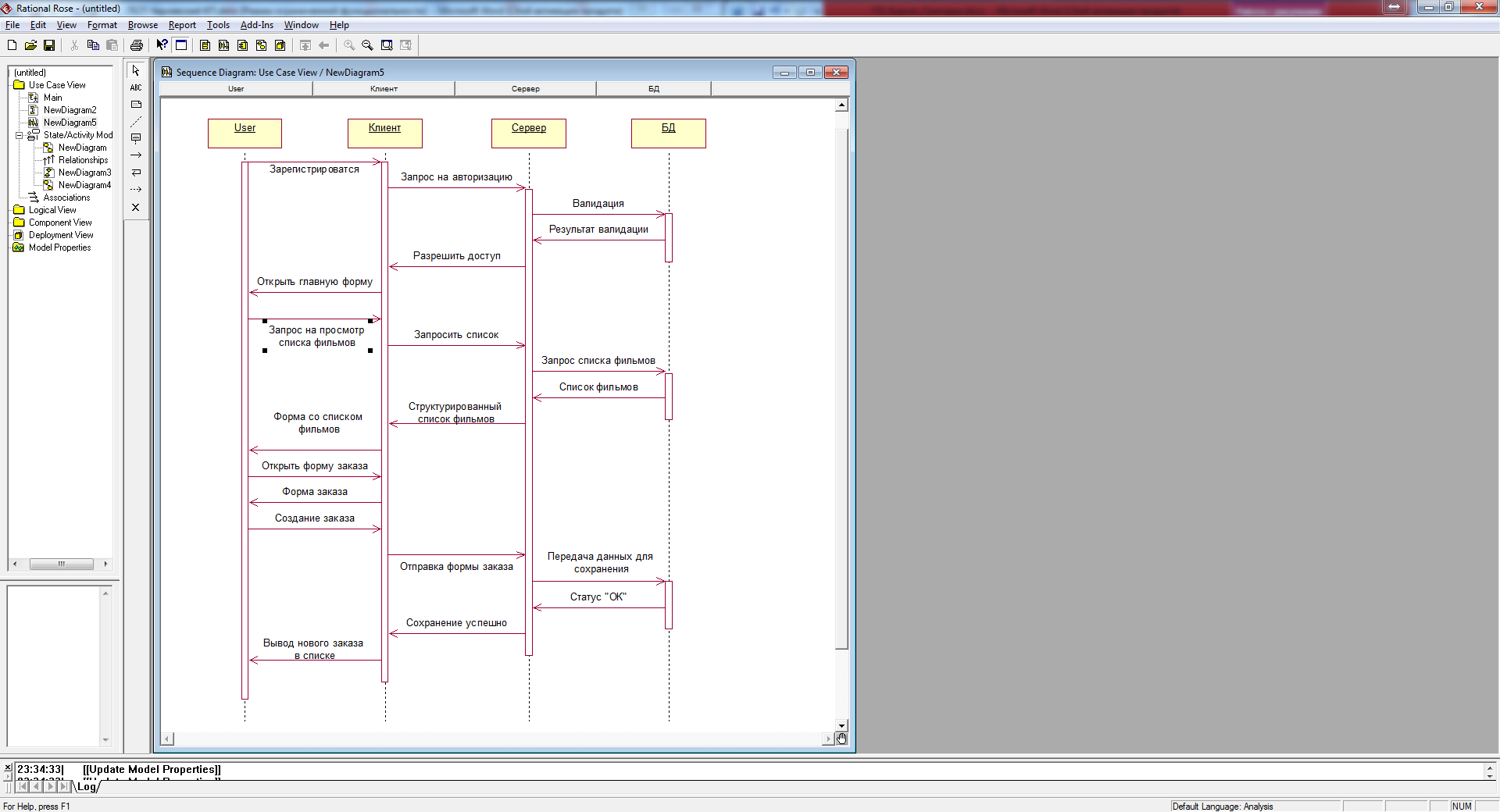


Рисунок Д.4 – Диаграмма последовательностей

# ПРИЛОЖЕНИЕ Е

# Листинг основных функций программы

(обязательное)

**public** **class** ***ConnectService*** **extends** ***UnicastRemoteObject*** **implements** IConnectService ***{***

**private** ICustomerRepository \_customerRepository;

**private** IOrderRepository \_orderRepository;

**private** IFilmRepository \_filmRepository;

**private** IEmployeeRepository \_employeeRepository;

**public** **ConnectService*()*** **throws** ***RemoteException*** ***{***

// thats bad, but i think it's better than call connectservice with

// those repositories

// otherwise we dont use IoC container here

**this**.\_customerRepository = **new** **CustomerRepository*()***;

**this**.\_orderRepository = **new** **OrderRepository*()***;

**this**.\_filmRepository = **new** **FilmRepository*()***;

**this**.\_employeeRepository = **new** **EmployeeRepository*()***;

***}***

**public** List<***Order***> **GetOrders*()*** ***{***

List<***Order***> **orderList** = **new** **ArrayList**<***Order***>***()***;

***Order*** **o1** = **new** **Order*()***;

**o1**.**setFilmId*(***1***)***;

**o1**.**setOrderId*(***2***)***;

**o1**.**setReturned*(*false*)***;

**orderList**.**add*(*o1*)***;

**return** **orderList**;

***}***

**public** List<***Film***> **GetFilmsList*()*** ***{***

**try** ***{***

**return** \_filmRepository.**GetAll*()***;

***}*** **catch** ***(Exception*** **e*)*** ***{***

**e**.**printStackTrace*()***;

***}***

**return** **null**;

***}***

**public** List<***CustomOrderView***> **GetCustomOrderViewList*()*** ***{***

**try** ***{***

**return** \_orderRepository.**GetCustomOrderView*()***;

***}*** **catch** ***(Exception*** **e*)*** ***{***

**e**.**printStackTrace*()***;

**return** **null**;

***}***

***}***

**public** **void** **AddNewCustomOrderView*(CustomOrderView*** **order*)*** **throws** ***RemoteException*** ***{***

***Customer*** **currentCustomer** = **null**;

**try** ***{***

***System***.***out***.**println*("Start Create order method")***;

**currentCustomer** = \_customerRepository.**GetByPassportNumber*(*order**.PassportNumber***)***;

**if** ***(*currentCustomer** == **null*)*** ***{***

// create new Customer

***Customer*** **cus** = **new** **Customer*()***;

**cus**.**setFirstName*(*order**.CustomerFirstName***)***;

**cus**.**setLastName*(*order**.CustomerLastName***)***;

**cus**.**setPassportNumber*(*order**.PassportNumber***)***;

**cus**.**setPhoneNumber*(*order**.PhoneNumber***)***;

\_customerRepository.**Create*(*cus*)***;

**currentCustomer** = \_customerRepository.**GetByPassportNumber*(*order**.PassportNumber***)***;

***}***

// get Customer with PassportNumber

**if** ***(*currentCustomer** == **null*)*** ***{***

***System***.***out***.**println*("Add customer's function didn't add any data!")***;

***}*** **else** ***{***

***Order*** **newOrder** = **new** **Order*()***;

**newOrder**.**setCustomerId*(*currentCustomer**.**getCustomerId*())***;

**newOrder**.**setCreated*(*order**.Created***)***;

**newOrder**.**setEmployeeId*(***0***)***;

**newOrder**.**setFilmId*(*order**.FilmId***)***;

**newOrder**.**setRentExpires*(*order**.RentExpires***)***;

**newOrder**.**setReturned*(*false*)***;

**newOrder**.**setCreated*(*new** **Date*())***;

***System***.***out***.**println*("Add new order - with that customerId:"*** + **currentCustomer**.**getCustomerId*())***;

\_orderRepository.**Create*(*newOrder*)***;

***}***

***}*** **catch** ***(Exception*** **e*)*** ***{***

// **TODO** Auto-generated catch block

**e**.**printStackTrace*()***;

**throw** **new** **RemoteException*("Sql add new order failed")***;

***}***

***}***

**public** **void** **UpdateOrderStatus*(*int** **orderId*)***

***{***

\_orderRepository.**UpdateReturnedStatusById*(*orderId**,1***)***;

***}***

**public** **boolean** **GetStatusConnect*()*** **throws** ***RemoteException*** ***{***

***System***.***out***.**println*("Check Status of the RMI Connect")***;

**return** **true**;

***}***

**public** **boolean** **IdentificationAccess*(String*** **login**, ***String*** **pass*)*** **throws** ***RemoteException*** ***{***

***System***.***out***.**println*("Try to authorize")***;

***Employee*** **empl**;

**try** ***{***

**empl** = \_employeeRepository.**GetByLoginAndPass*(*login**,**pass*)***;

**if** ***(*empl** == **null*)*** // not found user because have wrong login or password

***{***

***System***.***out***.**println*("login failed! Tried to gain access : "***+**login** ***)***;

**return** **false**;

***}***

***System***.***out***.**println*("Success login : "***+**empl**.**getFirstName*()***+ ***" "***+**empl**.**getLastName*()***+***" !"*** ***)***;

**return** **true**;

***}*** **catch** ***(Exception*** **e*)*** ***{***

**e**.**printStackTrace*()***;

**throw** **new** **RemoteException*("Sql Authorize failed")***;

***}***

***}***

***}***

***Работа с БД***

**public** **class** ***DataBaseProvider{***

**private** **static** Connection *Connect* = **null**;

**public** **static** Statement *Statement*;

**private** **static** ***DataBaseProvider*** *\_instance* = **null**;

**private** ***String*** uri = **null**;

**private** ***String*** login = **null**;

**private** ***String*** pass = **null**;

//constructor

**private** **DataBaseProvider*(){***

**InitConnection*()***;

***}***

**public** **static** Statement **GetNewStatement*()*** **throws** ***SQLException*** ***{***

**return** *Connect*.**createStatement*()***;

***}***

**private** **void** **InitConnection*()*** ***{***

**try*{***

**LoadDbConfig*()***;

**try** ***{***

***Class***.*forName****("com.mysql.jdbc.Driver")***;

***System***.***out***.**println*("Driver loading success!")***;

***}***

**catch** ***(ClassNotFoundException*** **e*)*** ***{***

***System***.***err***.**println** ***(*e*)*** ;

***System***.*exit* ***(***-1***)*** ;

***}***

//String url = "jdbc:mysql://localhost/";

*Connect* = ***DriverManager***.*getConnection****(***uri,login,pass***)***;

// create a statement

*Statement* = *Connect*.**createStatement*()***;

**boolean** **res** = *Statement*.**execute*("Select \* from fmdat.customer")***;

**if** ***(*res*)***

***System***.***out***.**println*("Connected to fmdat")***;

**else**

***System***.***out***.**println*("Something wrong! Check SQL connect")***;

***}***

**catch*(***java.sql.***SQLException*** **e*)***

***{***

**e**.**printStackTrace*()***;

***System***.***out***.**println*("\*\*\*\*\*\*\*\*\nConnect to MySql failed!\n\*\*\*\*\*\*\*\*\*\*")***;

***System***.*exit* ***(***-1***)*** ;

***}***

***}***

**public** **static** **synchronized** ***DataBaseProvider*** **GetInstance*()***

***{***

**if** ***(****\_instance* == **null*)***

*\_instance* = **new** **DataBaseProvider*()***;

**return** *\_instance*;

***}***

**private** **void** **LoadDbConfig*()*** ***{***

***FileInputStream*** **fis**;

***Properties*** **property** = **new** **Properties*()***;

**try** ***{***

**fis** = **new** **FileInputStream*("src/config.cfg")***;

// load config.sys from stream

**property**.**load*(*fis*)***;

uri = **property**.**getProperty*("db.uri")***;

login = **property**.**getProperty*("db.login")***;

pass = **property**.**getProperty*("db.password")***;

***System***.***out***.**println*("CHECK->URI: "*** + uri

+ ***", LOGIN: "*** + login

+ ***", PASSWORD: "*** + pass***)***;

***}*** **catch** ***(IOException*** **e*)*** ***{***

***System***.***err***.**println*("ОШИБКА: Файл config.sys отсуствует или его невозможно открыть!")***;

***}***

***}***

@Override

**protected** **void** **finalize*()*** **throws** ***Throwable***

***{***

//close connection when exit application

*Connect*.**close*()***;

**super**.**finalize*()***;

***}***

***}***

***Инициализация главного окна сервера и регистрация RMI***

**public** **static** **void** **main*(String[]*** **args*)*** ***{***

***EventQueue***.*invokeLater****(*new** **Runnable*()*** ***{***

**public** **void** **run*()*** ***{***

**try** ***{***

***Form*** **window** = **new** **Form*()***;

**window**.frmServerside.**setVisible*(*true*)***;

***DataBaseProvider***.*GetInstance****()***;

//new Thread (new SimpleServerSocket());

***System***.*setProperty****("java.rmi.server.hostname"***,***"127.0.0.1")***;

***LocateRegistry***.*createRegistry****(***1099***)***;

IConnectService **serv** = **new** **ConnectService*()***;

***Naming***.*rebind****("ConnectService"***, **serv*)***;

***String[]*** **serversList** = ***Naming***.*list****("127.0.0.1")***;

// debug

**for** ***(String*** **item** : **serversList*)*** ***{***

***System***.***out***.**println*(*item*)***;

***}***

**if** ***(*serversList**.length>0***)*** // if we have registered out server - tell about that

***System***.***out***.**println*("RMI Connected")***;

***}*** **catch** ***(Exception*** **e*)*** ***{***

**e**.**printStackTrace*()***;

***}***

***}***

***})***;

Пример защиты ввода значений в текстовое поле на клиенте

tf\_rentDays.**addKeyListener*(*new** java.awt.event.**KeyAdapter*()*** ***{***

**public** **void** **keyTyped*(KeyEvent*** **e*)*** ***{***

**char** **key** = **e**.**getKeyChar*()***;

// ignore all no digit

**if** ***(***!***Character***.*isDigit****(*key*))***

**e**.**consume*()***;

***}***

**public** **void** **keyReleased*(KeyEvent*** **e*){***

**if** ***(***tf\_rentDays.**getText*()***.**length*()***>0***)***

***{***

**int** **i** = filmComboBox.**getSelectedIndex*()***;

**double** **newValue** = **data**.**get*(*i*)***.**getRentCost*()*** \*

***Integer***.*parseInt****(***tf\_rentDays.**getText*())***;

**lbl\_Payment**.**setText*(Double***.*toString****(*newValue*))***;

***}***

**else*{***

**lbl\_Payment**.**setText*("0")***;

***}***

***}***

***})***;