МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И ПРОГРАММИРОВАНИЯ

КУРСОВАЯ РАБОТА   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ст. преп. |  |  |  | Е.О. Пятлина |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ |
| ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ СРЕДСТВАМИ ЯЗЫКА UML |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4631 |  |  |  | С.А. Гришин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2018

**Содержание**

ВВЕДЕНИЕ3

1. Описание основных функций Информационной системы «База данных о фильмах»3
2. Краткая информация об аппарате объектно-ориентированного проектирования программного обеспечения информационной систем 4
3. CASE средства, используемые для проектирования ПО ИС, их возможности, достоинства, особенности применения.4
4. Разработка ПО информационной системы «База данных о фильмах».5
   1. Use-case диаграмма 5
   2. Диаграмма классов7
   3. Диаграммы последовательностей10
   4. Диаграммы состояний16
   5. Диаграммы видов деятельности5
   6. Диаграмма размещения 5
   7. Диаграмма пакетов5

ЗАКЛЮЧЕНИЕ5

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ5

ПРИЛОЖЕНИЕ5

**Введение**

В данной работе приведен проект построения информационной системы. В качестве предметной области рассмотрена «База данных о фильмах». Проектирование производилось с помощью специализированного программного обеспечения Software Ideas Modeler – программного пакета для создания диаграмм UML. Приложение полностью совместимо с UML 2.2 и включает всевозможные типы диаграмм, используемых в этой системе.

**1. Описание основных функций Информационной системы «База данных о фильмах»**

Информационная система «База данных о фильмах» применяется для предоставления пользователям информации о фильмах. Она позволяет просматривать всю информацию о фильме и делиться своим мнением о нём с другими пользователями.

* Работа администратора с базой данных
  + Добавление новых фильмов в базу данных
  + Обновление информации о фильмах
  + Редактирование информации о фильмах
    - Описания
    - Постеров
    - Рецензий
    - Комментариев
    - Видео
* Регистрация нового пользователя
  + Регистрация по электронной почте
  + Регистрация по номеру телефона
  + Регистрация с помощью соц. Сетей
  + Восстановление доступа
* Добавление информации к фильму
  + Добавление комментария к фильму
  + Добавление рецензии к фильму
* Информация о фильме
  + Чтение рецензий
  + Чтение комментриев
  + Просмотр постеров
  + Просмотр фильма
* Поиск фильма по категории

**2. Краткая информация об аппарате объектно-ориентированного проектирования программного обеспечения информационной систем**

**UML** (англ. Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур. UML является языком широкого профиля, это — открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью.

UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода. Словарь языка UML включает три вида строительных блоков: сущности, отношения, диаграммы.

**Сущности в UML** — это абстракции, являющиеся основными элементами модели. Отношения связывают различные сущности; диаграммы группируют представляющие интерес совокупности сущностей.

В UML имеется четыре типа сущностей: структурные, поведенческие, группирующие, аннотационные.

В языке UML определены **четыре типа отношений**: зависимость, ассоциация, обобщение, реализация.

**Диаграмма в UML** — это графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграммы рисуют для визуализации системы с разных точек зрения.

Теоретически диаграммы могут содержать любые комбинации сущностей и отношений. На практике, однако, применяется сравнительно небольшое количество типовых комбинаций, соответствующих пяти наиболее употребительным видам, которые составляют архитектуру ИС.

**3.CASE средства, используемые для проектирования ПО ИС, их возможности, достоинства, особенности применения.**

CASE средства, поддерживающие методологию UML:

* Rational Rose (Rational Software)
* Paradigm Plus (CA/Platinum)
* ARIS (IDS Sheer AG)
* Together Designer (Borland)
* и др.

Система Rational Rose позволяет строить восемь типов диаграмм UML: диаграммы прецедентов, диаграммы классов, диаграммы последовательностей, диаграммы кооперации, диаграммы состояний, диаграммы действий, компонентные диаграммы, диаграммы развертывания. Основным типом диаграмм, своеобразным ядром моделирования в UML являются диаграммы классов. Кроме UML, предусмотрено использование и других методов (Booch, OMT).

Система Paradigm Plus ориентирована на методологию OOCL (Object Oriented Change and Learning) и компонентную технологию проектирования и разработки. Она поддерживает диаграммы различных методов (UML, CLIPP, TeamFusion, OMT, Booch, OOCL, Martin/Odell, Shlaer/Mellor, Coad/Yourdon).

Система ARIS обеспечивает четыре различных «взгляда» на моделирование и анализ: Процессы, Функции (с Целями), Данные, Организация. Для каждого «взгляда» поддерживаются три уровня анализа (требования, спецификации, внедрение). Каждый из уровней анализа состоит из своего комплекта моделей различных типов, в том числе диаграмм UML, диаграмм SAP/R3 и др.

На этапе создания клиентского и серверного кода все современные средства UML-моделирования могут осуществлять генерацию кода на различных языках программирования.

При проектирование информационной системы была использована система Software Ideas Modeler позволяет строить все типы диаграмм UML: диаграммы прецедентов, диаграммы классов, диаграммы последовательностей, диаграммы кооперации, диаграммы состояний, диаграммы действий, компонентные диаграммы, диаграммы развертывания. Выбранное CASE выполняет все требования решаемой задачи.

**4. Разработка ПО информационные системы «База данных о фильмах»**

**4.1. Use-case диаграмма**

Диаграммы позволяют наглядно представить ожидаемое поведение системы.

В созданной диаграмме использовались все возможные виды связей между элементами диаграммы:

* коммуникация (communication),
* включение (include),
* расширение (extend),
* обобщение (generalization).

Диаграммы Use Case по концепции напоминают классические DFD диаграммы, также применяемые для структурного анализа. Use Case также отображают границы исследуемой системы, её функциональность и определяют сущности и процессы, а также пользователей системы. При составлении этих диаграмм существует некоторая неопределённость.

Она состоит в проблеме определения различий между сущностями и пользователями системы, а также её администраторами. Диаграммы Use Case чаще всего используются на этапе трансформации логической архитектуры системы в концептуальную модель, реализуемую при объектном подходе с определением событийной структуры управления будущим программным проектом.

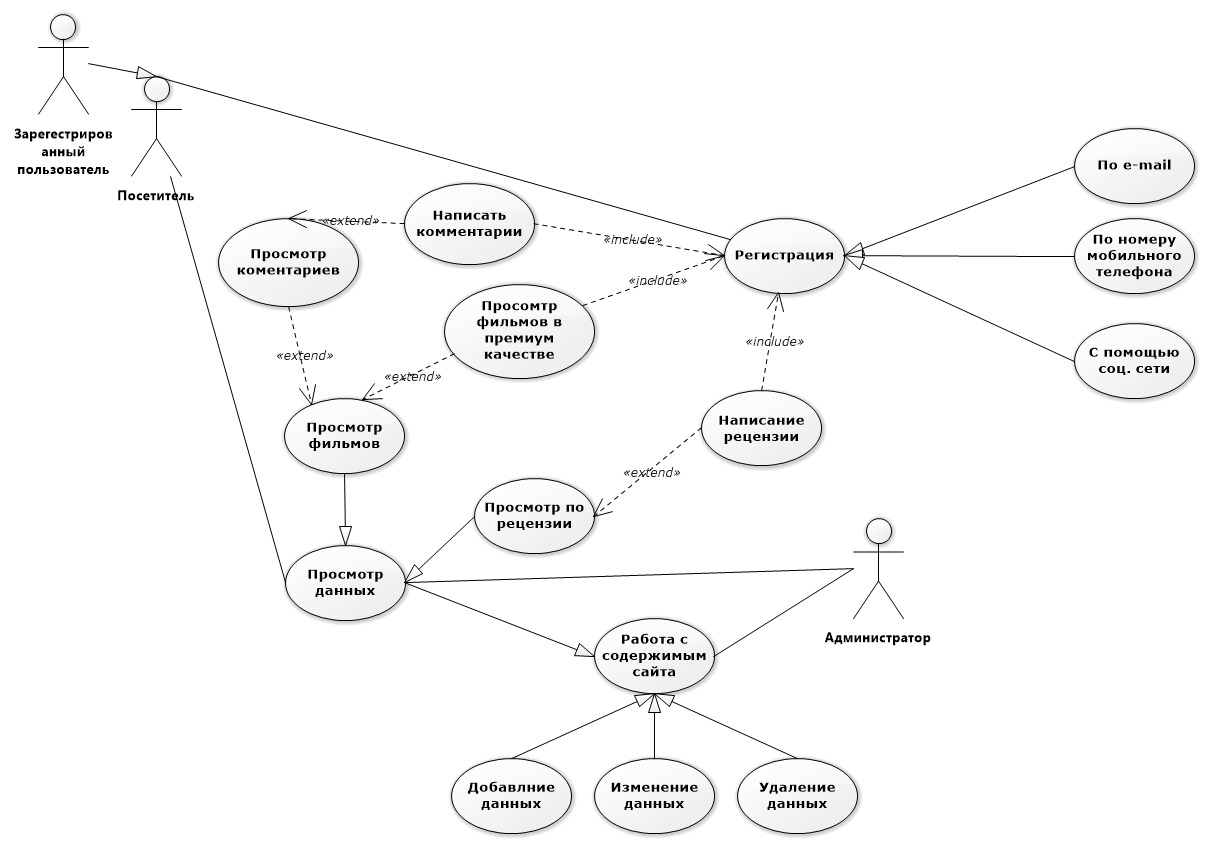
****

Рис.1 Общая USE-CASE диаграмма

**Расчет оценки диаграммы**

****

где *Sobj*-оценка элемента на диаграмме, *Slink*- оценка связей, *Оbj*- кол-во объектов на диаграмме, *Tobj*–количество типов объектов, *Tlink*- количество типов связи.

Значение немного выше рекомендованного, близка к указанному диапазону.

**4.2. Диаграмма классов**

[**Диаграмма классов**](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2) (Class diagram) — статическая структурная диаграмма, описывающая структуру системы, она демонстрирует классы системы, их атрибуты, методы и зависимости между классами. Существуют разные точки зрения на построение диаграмм классов в зависимости от целей их применения:

* концептуальная точка зрения — диаграмма классов описывает модель предметной области, в ней присутствуют только классы прикладных объектов;
* точка зрения спецификации — диаграмма классов применяется при проектировании информационных систем;
* точка зрения реализации — диаграмма классов содержит классы, используемые непосредственно в программном коде (при использовании объектно-ориентированных языков программирования).

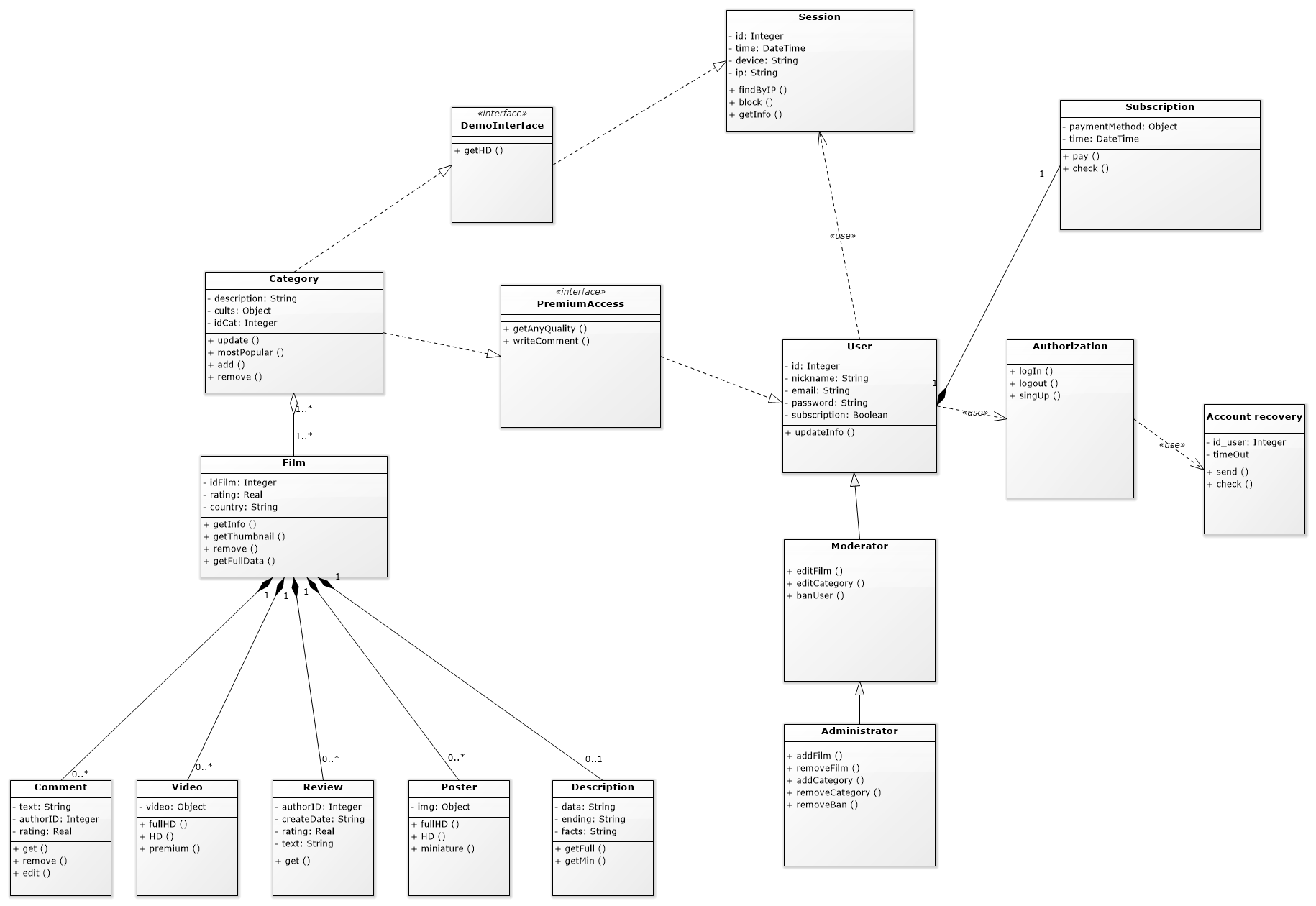
****

Рис. 2 Диаграмма классов.

**Расчет оценки диаграммы**

где *Sobj*-оценка элемента на диаграмме, *Slink* - оценка связей, *Оbj* - кол-во объектов на диаграмме, *Tobj*–количество типов объектов, *Tlink*- количество типов связи.

где *Op*- количество операций, *Atr*- количество атрибутов.

**Расчет оценки диаграммы классов с атрибутами и операциями**

**4.3. Диаграммы последовательностей**

Это [диаграмма](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), на которой показаны взаимодействия объектов, упорядоченные по времени их проявления. Основными элементами диаграммы последовательностей являются обозначения объектов (прямоугольники), вертикальные линии, отображающие течение времени при деятельности объекта, и стрелки, показывающие выполнение действий объектами.

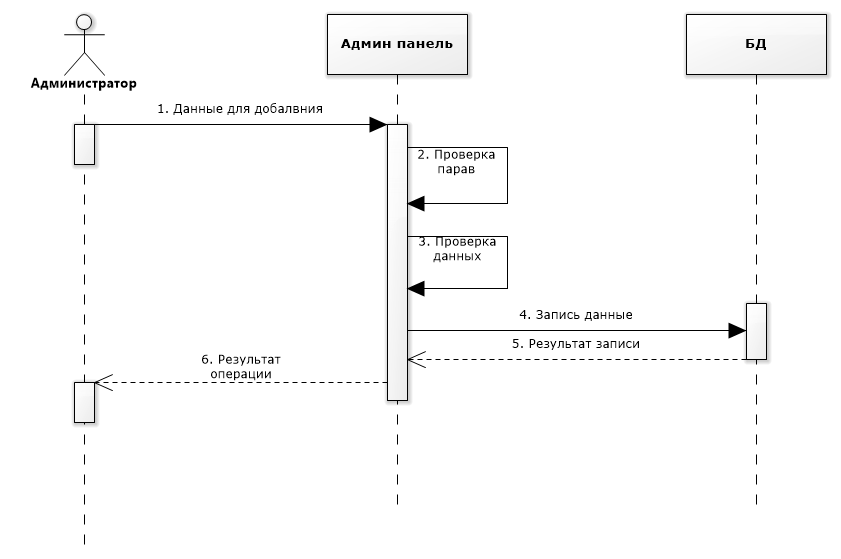


Рис.3 Диаграмма последовательностей для сценария «Добавление фильма».

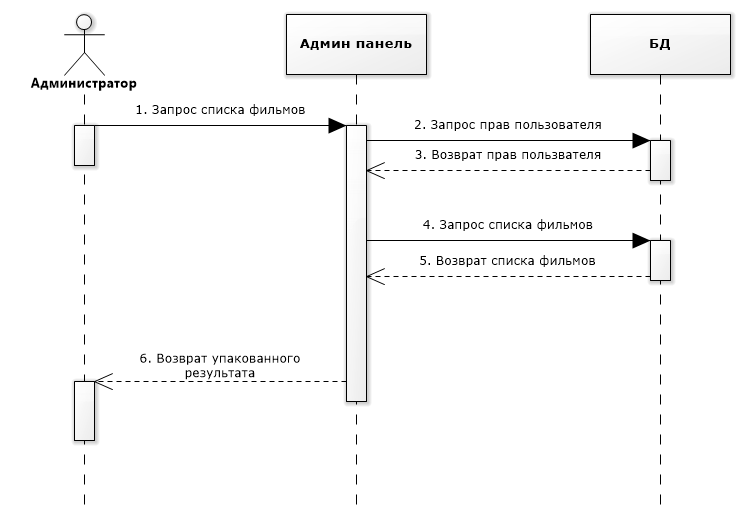


Рис.4 Диаграмма последовательностей для сценария «Получение списка фильмов в панели управления».

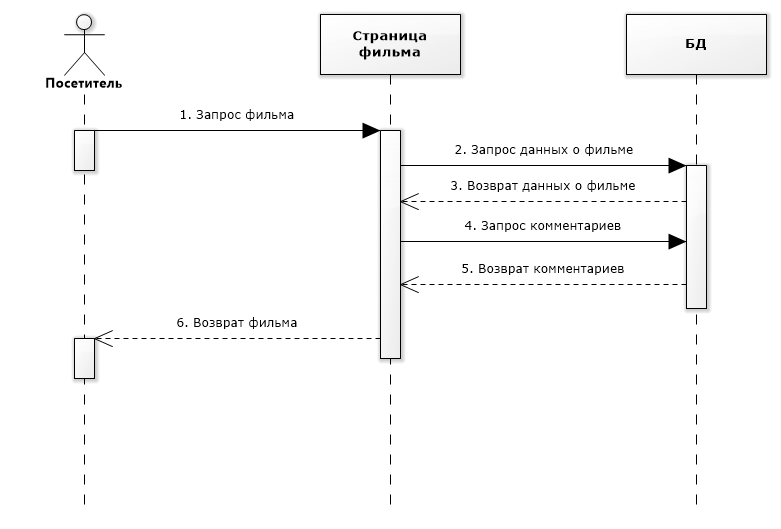


Рис.5 Диаграмма последовательностей для сценария «Получения списка фильмов пользователем».

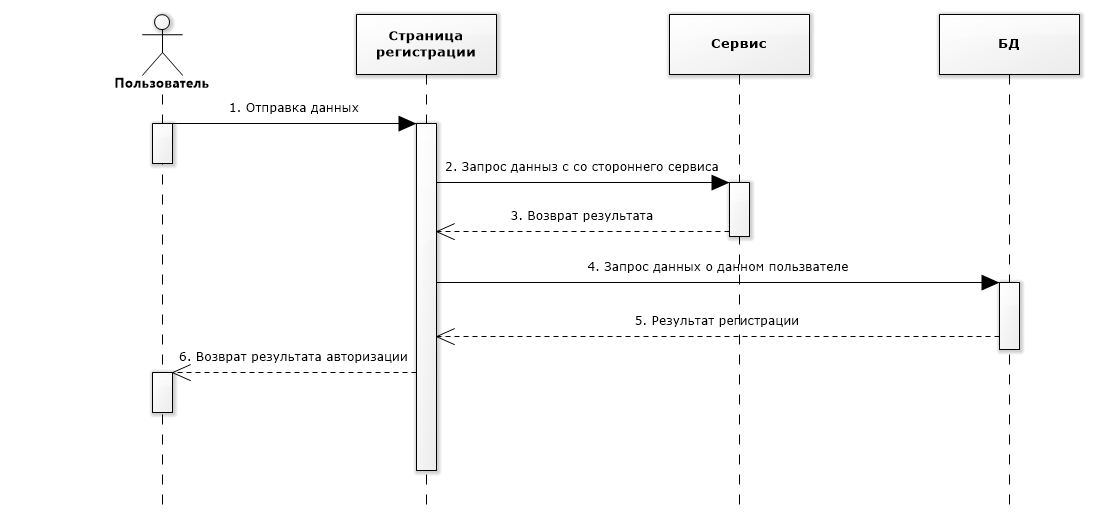


Рис.6 Диаграмма последовательностей для сценария «Регистрация пользователя по с помощью соц. сети».

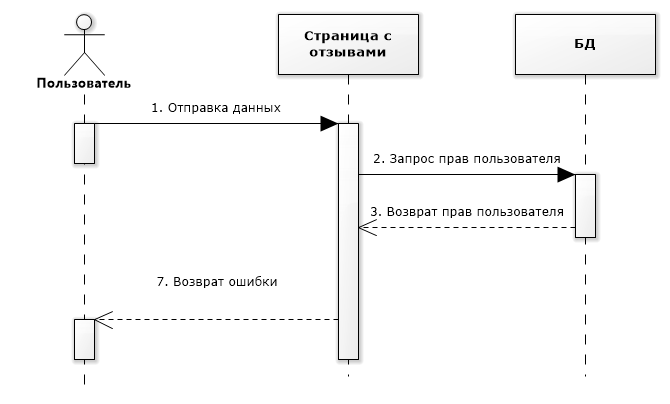


Рис.7 Диаграмма последовательностей для сценария «Оставить отзыв о фильме не авторизованным пользователем».

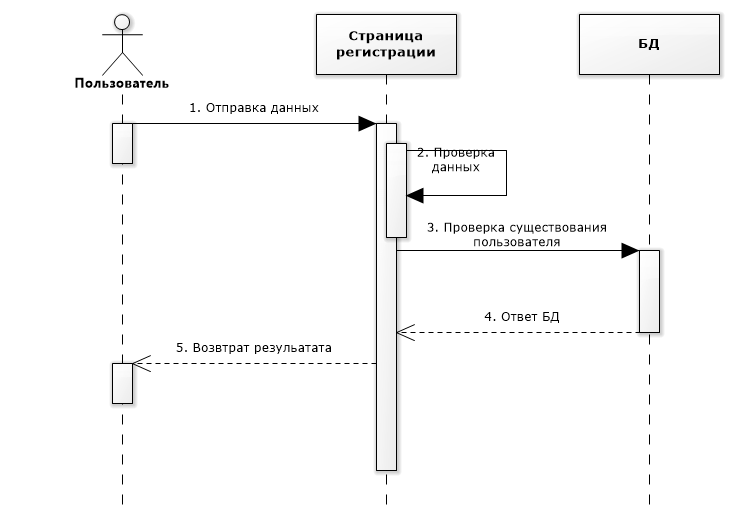


Рис.8 Диаграмма последовательностей для сценария «Авторизация пользователя».

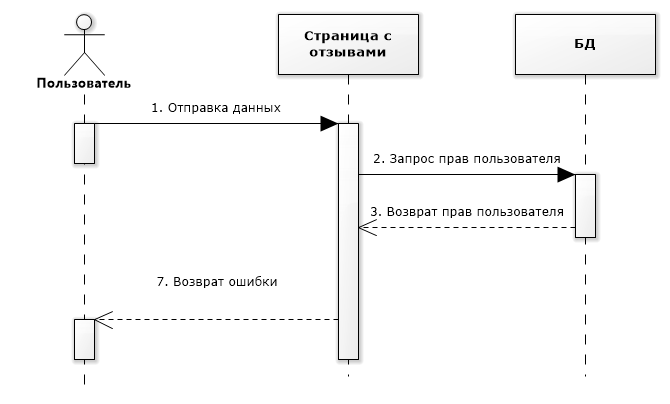


Рис.9 Диаграмма последовательностей для сценария «Написание отзыва не авторизованным пользователем».

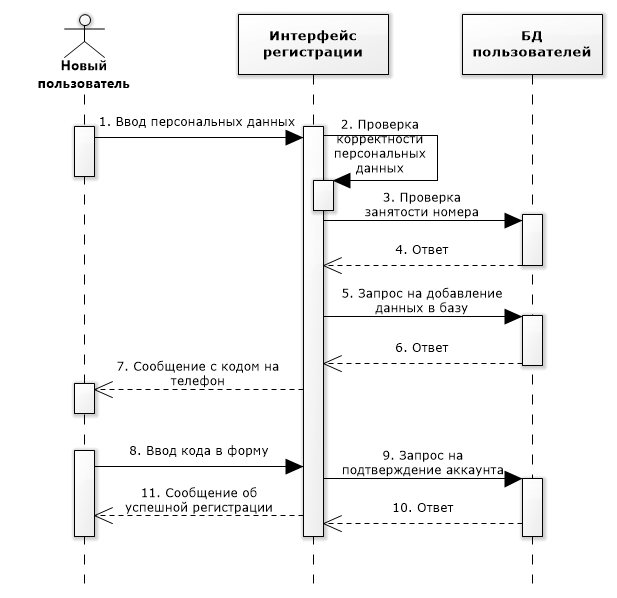
****

Рис.10 Диаграмма последовательностей для альтернативного сценария «Регистрация с помощью СМС».

**4.4. Диаграммы состояний (Statechar diagram)**

Диаграмма состояний (Statechar diagram) определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий.

Существует много форм диаграмм состояний, незначительно отличающихся друг от друга семантикой.

На диаграмме имеются два специальных состояния – начальное (start) и конечное(stop). Начальное состояние выделено черной точкой, оно соответствует состоянию объекта, когда он только что был создан. Конечное состояние обозначается черной точкой в белом кружке, оно соответствует состоянию объекта непосредственно перед его уничтожением. Когда объект находится в каком-то конкретном состоянии, могут выполняться различные процессы. Процессы, происходящие, когда объект находится в определенном состоянии, называются действиями (actions).

ЗАКОНЧИЛ ТУТ



Рис 11. Диаграмма состояний для класса «User».

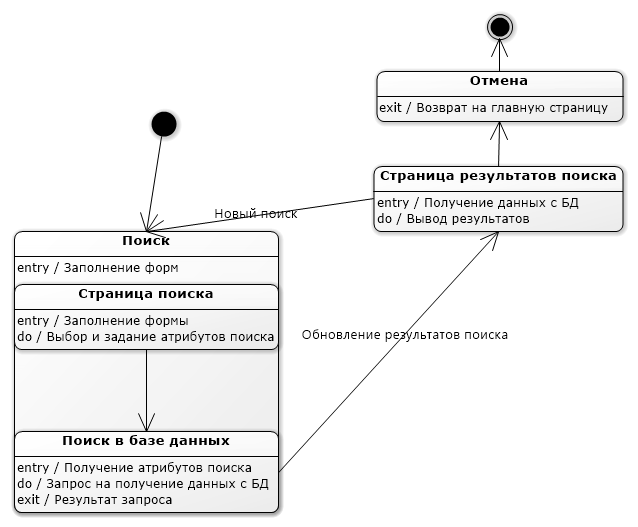


Рис 12. Диаграмма состояний для класса «Search».

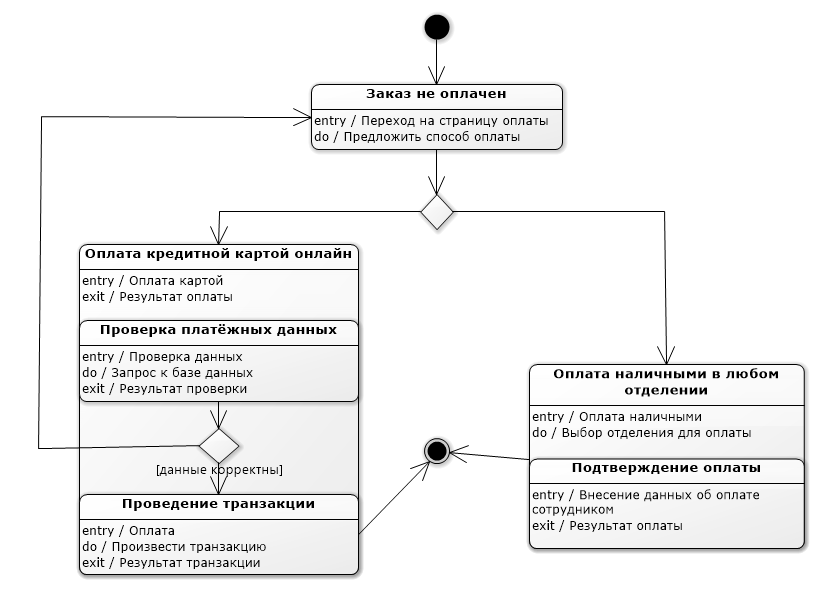


Рис 13. Диаграмма состояний для класса «PaymentOrder».

Рис 14. Диаграмма состояний для класса «CreditCard».

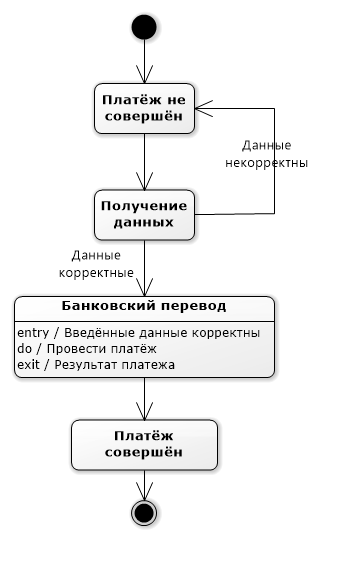


Рис 15. Диаграмма состояний для класса «Cash».

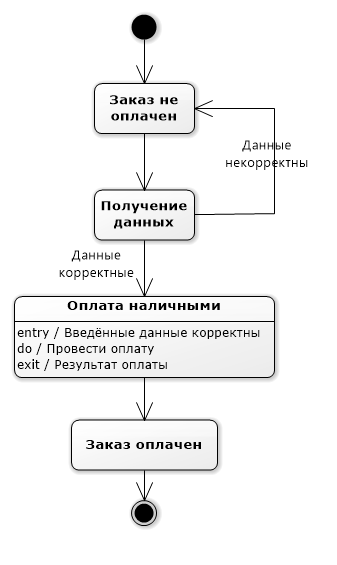


Рис 16. Диаграмма состояний для класса «Sale».



Рис 17. Диаграмма состояний для класса «Tour».

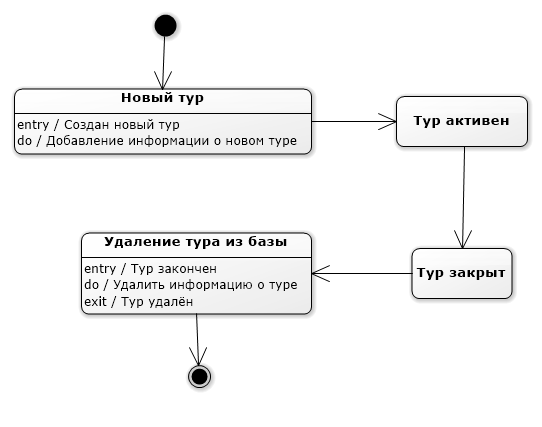
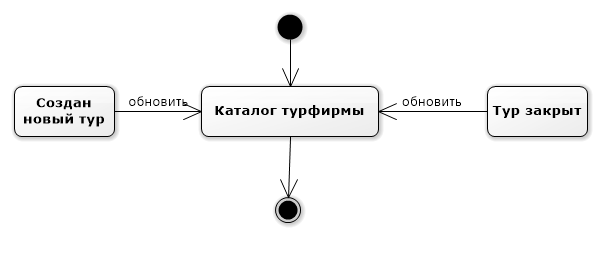
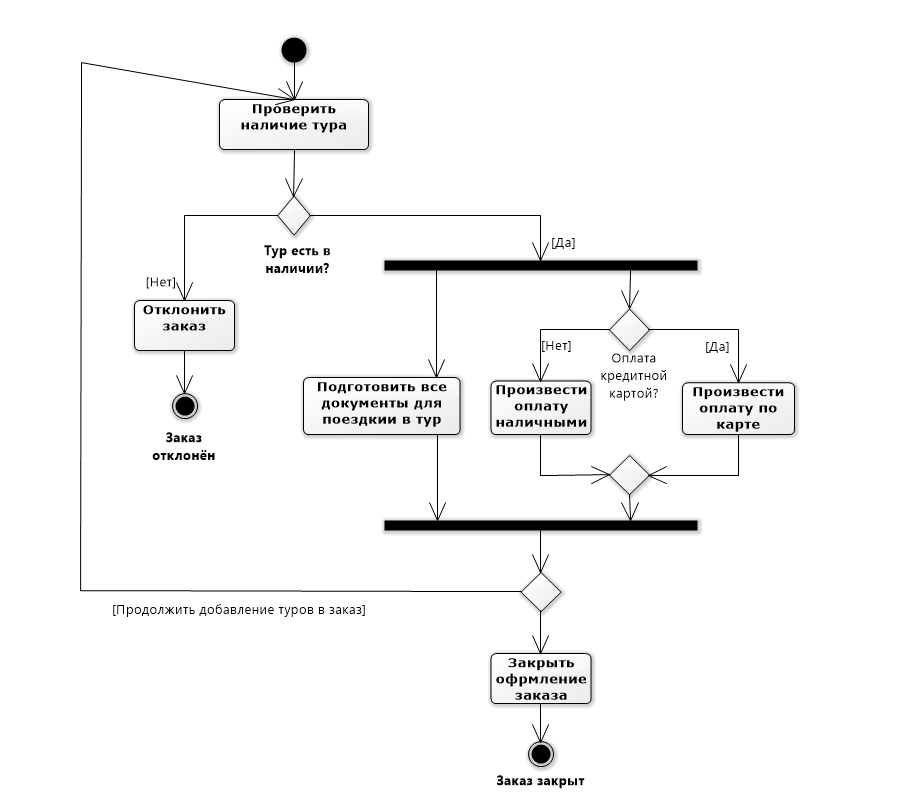


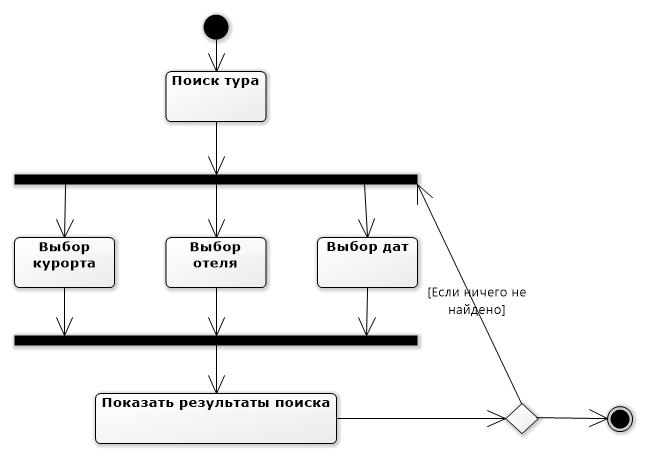
Рис 18. Диаграмма состояний для класса «TourCatalog».

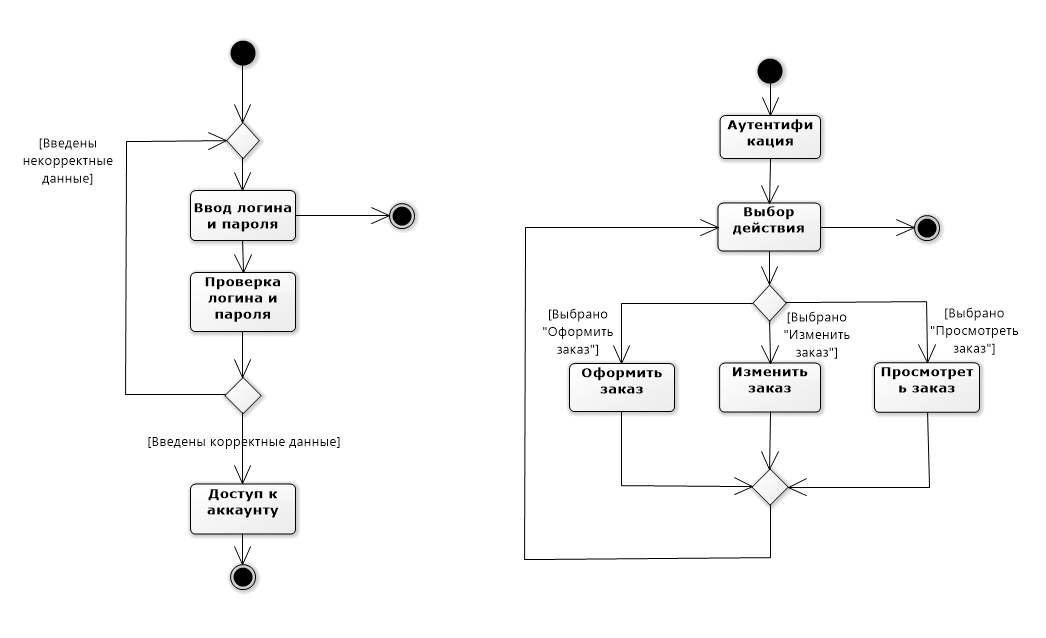


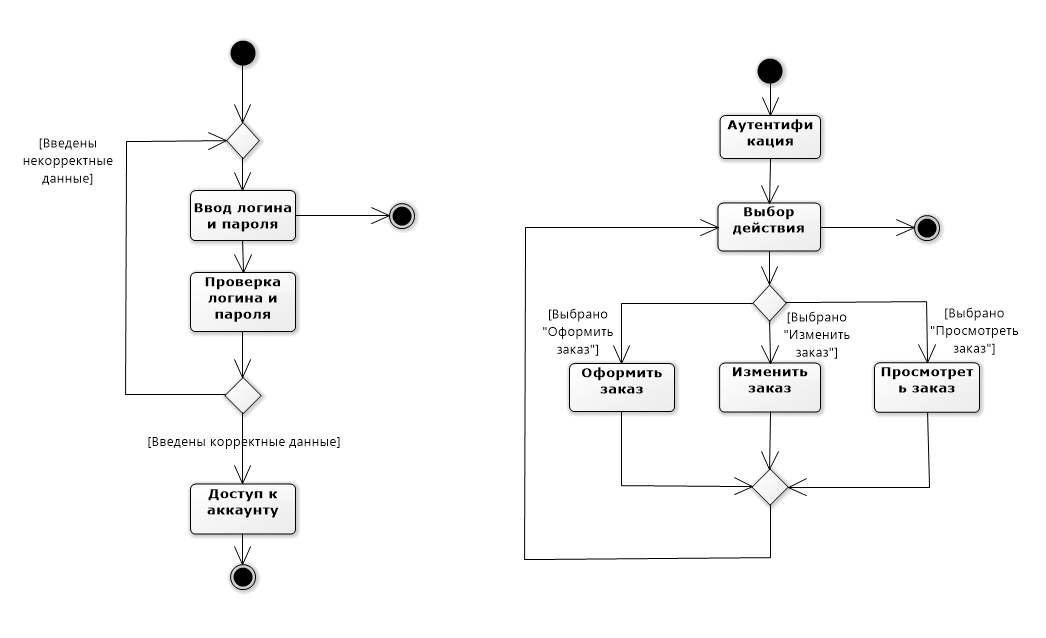
**4.5 Диаграммы видов деятельности(Activity diagram)**

Диаграммы видов деятельности(Activity diagram)-диаграмма, на которой показано разложение некоторой деятельности на ее составные части. Под деятельностью понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчиненных элементов - вложенных видов деятельности и отдельных действий, соединенных между собой потоками, которые идут от выходов одного узла ко входам другого.

  
Рис 19. Диаграмма деятельности для метода «CreateOrder» класса «Order».

  
Рис 20. Диаграмма деятельности для метода «Search» класса «TourCatalog».

  
Рис 21. Диаграмма деятельности для метода «Login» класса «Customer».

  
Рис 22. Диаграмма деятельности для метода «ChangeOrder» класса «Order».

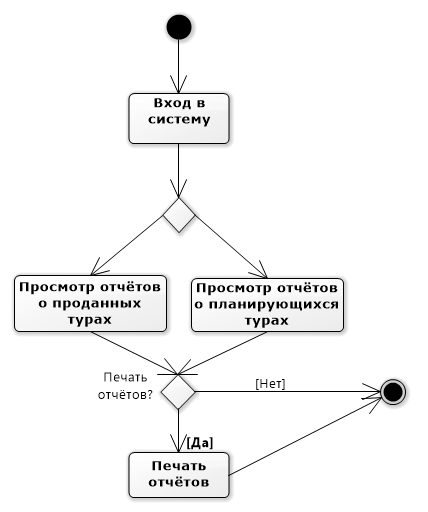
  
Рис 23. Диаграмма деятельности для метода «GetInformationAboutTours» класса «TourReport».

Рис 24. Диаграмма деятельности для метода «MakePayment» класса «PaymentMethod».

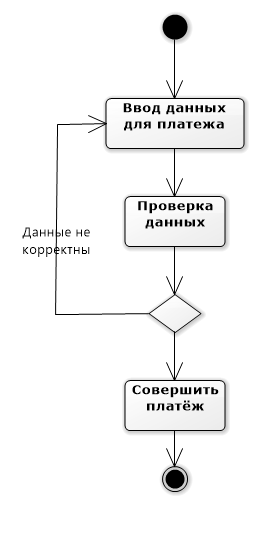


Рис 25. Диаграмма деятельности для метода «ViewListTours» класса «TourAgency».



Рис 26. Диаграмма деятельности для метода «CalculateSale» класса «Sale».

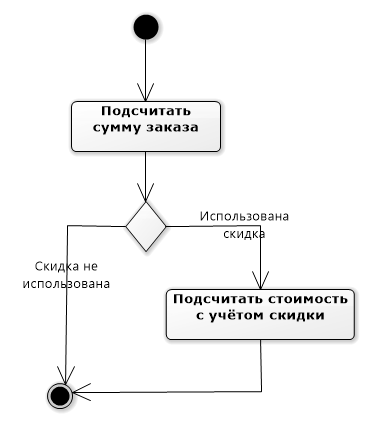
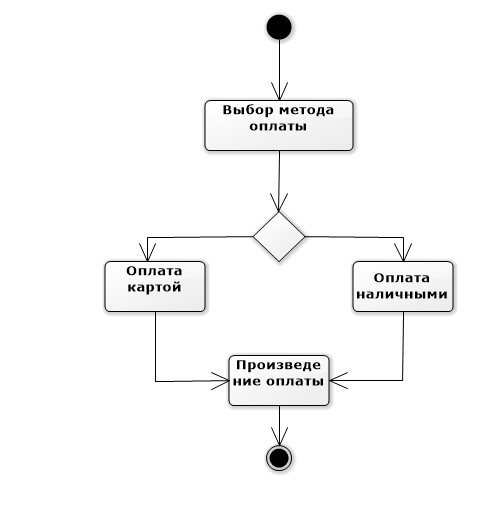
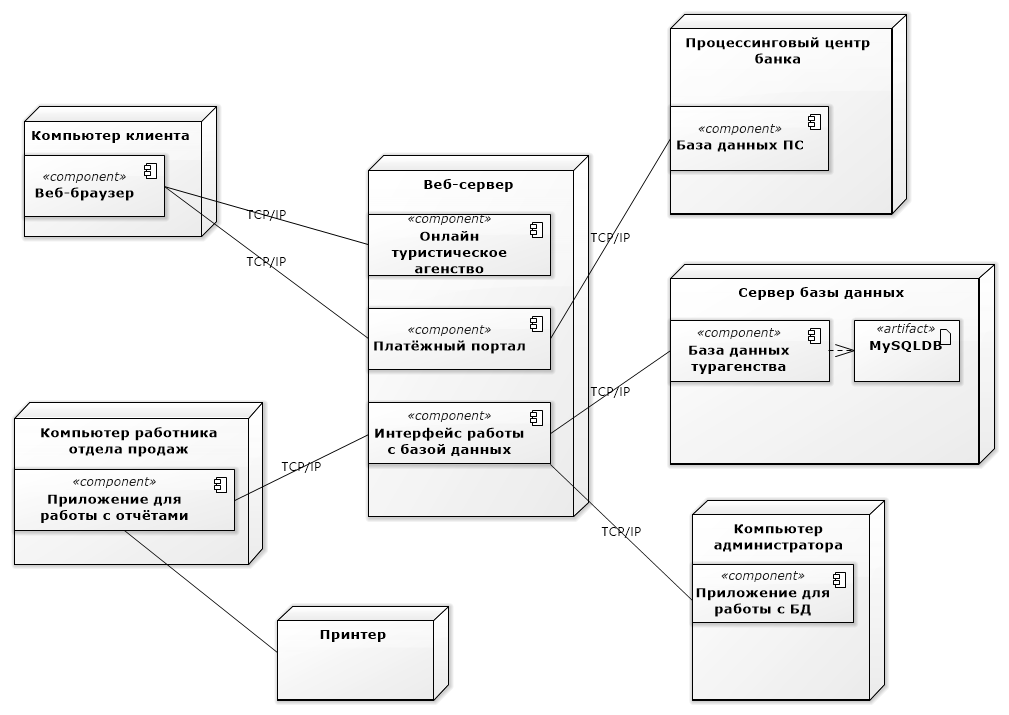


Рис 27. Диаграмма деятельности для метода «SelectPaymentMethod» класса «Customer».



**4.6.Диаграмма размещений (Диаграмма развертывания).**

Диаграмма развертывания (Deployment diagram)- предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения. Диаграмма развертывания отражает физические взаимосвязи между программными и аппаратными компонентами разрабатываемой системы. Каждый узел на диаграмме развертывания представляет собой некоторый тип вычислительного устройства - в большинстве случаев самостоятельную часть аппаратуры.



**ЗАКЛЮЧЕНИЕ**

В данной курсовой работе спроектирована информационная система «База данных о фильмах» в нотации UML с использованием Сase-средства Ideas Software Modeler.

Система описана практически со всех возможных точек зрения, рассмотрены разные аспекты поведения системы, диаграммы сравнительно просты для чтения, методы описания результатов анализа и проектирования семантически близки к методам программирования на современных ОО-языках.

Данная работа дает возможность организовать качественное функционирование описанной информационной системы, позволяет автоматизировать процессы и организовать поддерживаемый сервис.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Хассан Гома: UML Проектирование систем реального времени, распределенных и параллельных приложений – ДМК-Пресс, 2016. – 700c
2. Леоненков А.В. Самоучитель UML: монография/ А. Леоненков. - СПб. и др.: BHV - Санкт-Петербург, 2014 - 298 с.
3. Буч Г., Якобсон А., Рамбо Дж. Язык UML. Руководство пользователя / Пер. с англ. — ДМК Пресс , 2014 г.
4. Software Ideas Modeler - Agile CASE tool for software design & analysis [Электронный ресурс] <https://www.softwareideas.net/>
5. Пайлон, Питмен: UML 2 для программистов – Питер, 2012 – 240c.

**Приложение: Результаты автоматической генерации кода программ на основе разработанной диаграммы классов**

class AccountRecovery

{

private:

int id\_user;

object timeOut;

public:

void send ();

void check ();

};

void AccountRecovery::send ()

{

}

void AccountRecovery::check ()

{

}

class Session

{

private:

int id;

DateTime time;

string device;

string ip;

public:

void findByIP ();

void block ();

void getInfo ();

};

void Session::findByIP ()

{

}

void Session::block ()

{

}

void Session::getInfo ()

{

}

class Review

{

private:

int authorID;

string createDate;

float rating;

string text;

public:

void get ();

};

void Review::get ()

{

}

class Comment

{

private:

string text;

int authorID;

float rating;

public:

void get ();

void remove ();

void edit ();

};

void Comment::get ()

{

}

void Comment::remove ()

{

}

void Comment::edit ()

{

}

class Poster

{

private:

Object img;

public:

void fullHD ();

void HD ();

void miniature ();

};

void Poster::fullHD ()

{

}

void Poster::HD ()

{

}

void Poster::miniature ()

{

}

class Film

{

private:

int idFilm;

float rating;

string country;

public:

void getInfo ();

void getThumbnail ();

void remove ();

void getFullData ();

};

void Film::getInfo ()

{

}

void Film::getThumbnail ()

{

}

void Film::remove ()

{

}

void Film::getFullData ()

{

}

class Description

{

private:

string data;

string ending;

string facts;

public:

void getFull ();

void getMin ();

};

void Description::getFull ()

{

}

void Description::getMin ()

{

}

class Authorization

{

public:

void logIn ();

void logout ();

void singUp ();

};

void Authorization::logIn ()

{

}

void Authorization::logout ()

{

}

void Authorization::singUp ()

{

}

class Subscription

{

private:

Object paymentMethod;

DateTime time;

public:

void pay ();

void check ();

};

void Subscription::pay ()

{

}

void Subscription::check ()

{

}

class Video

{

private:

Object video;

public:

void fullHD ();

void HD ();

void premium ();

};

void Video::fullHD ()

{

}

void Video::HD ()

{

}

void Video::premium ()

{

}

class Moderator : public User

{

public:

void editFilm ();

void editСategory ();

void banUser ();

};

void Moderator::editFilm ()

{

}

void Moderator::editСategory ()

{

}

void Moderator::banUser ()

{

}

class Administrator : public Moderator

{

public:

void addFilm ();

void removeFilm ();

void addСategory ();

void removeСategory ();

void removeBan ();

};

void Administrator::addFilm ()

{

}

void Administrator::removeFilm ()

{

}

void Administrator::addСategory ()

{

}

void Administrator::removeСategory ()

{

}

void Administrator::removeBan ()

{

}

class User

{

private:

int id;

string nickname;

string email;

string password;

bool subscription;

public:

void updateInfo ();

};

void User::updateInfo ()

{

}

class Category

{

private:

string description;

Object cults;

int idCat;

public:

void update ();

void mostPopular ();

void add ();

void remove ();

};

void Category::update ()

{

}

void Category::mostPopular ()

{

}

void Category::add ()

{

}

void Category::remove ()

{

}

class PremiumAccess :

{

public:

void getAnyQuality ();

void writeComment ();

};

void PremiumAccess::getAnyQuality ()

{

}

void PremiumAccess::writeComment ()

{

}

class DemoInterface :

{

public:

void getHD ();

};

void DemoInterface::getHD ()

{

}