МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

КУРСОВОЙ ПРОЕКТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Е.О. Пятлина |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ | | | | |
| ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ «АГЕНСТВО НЕДВИЖИМОСТИ» СРЕДСТВАМИ ЯЗЫКА UML | | | | |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ | | | | |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТКА ГР.№ | 4831 |  |  |  | К.А. Корнющенков |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2020

# Место для оглавления

# ВВЕДЕНИЕ

В процессе создания крупных информационных систем перед разработчиками может возникнуть множество разнообразных проблем, которые возникают из-за ресурсных и других ограничений. Эти барьеры не всегда можно преодолеть классическими средствами структурного подхода, гораздо удобнее будет использование обектно-ориентированных методов. Такие программые средства моделирования, как Rational Rose позволяют уменьшить временные затраты на разработку информационной системы, в следствие чего сокращаются и другие издержки.

Целью выполнения данной курсовой работы является разработка информационной системы «Книжный магазин», которая отражала бы работу реального предприятия и позволила бы упростить управление им.

## Проектирование программного обеспечения информационной системы «Агентство недвижимости»

### Диаграмма вариантов использования

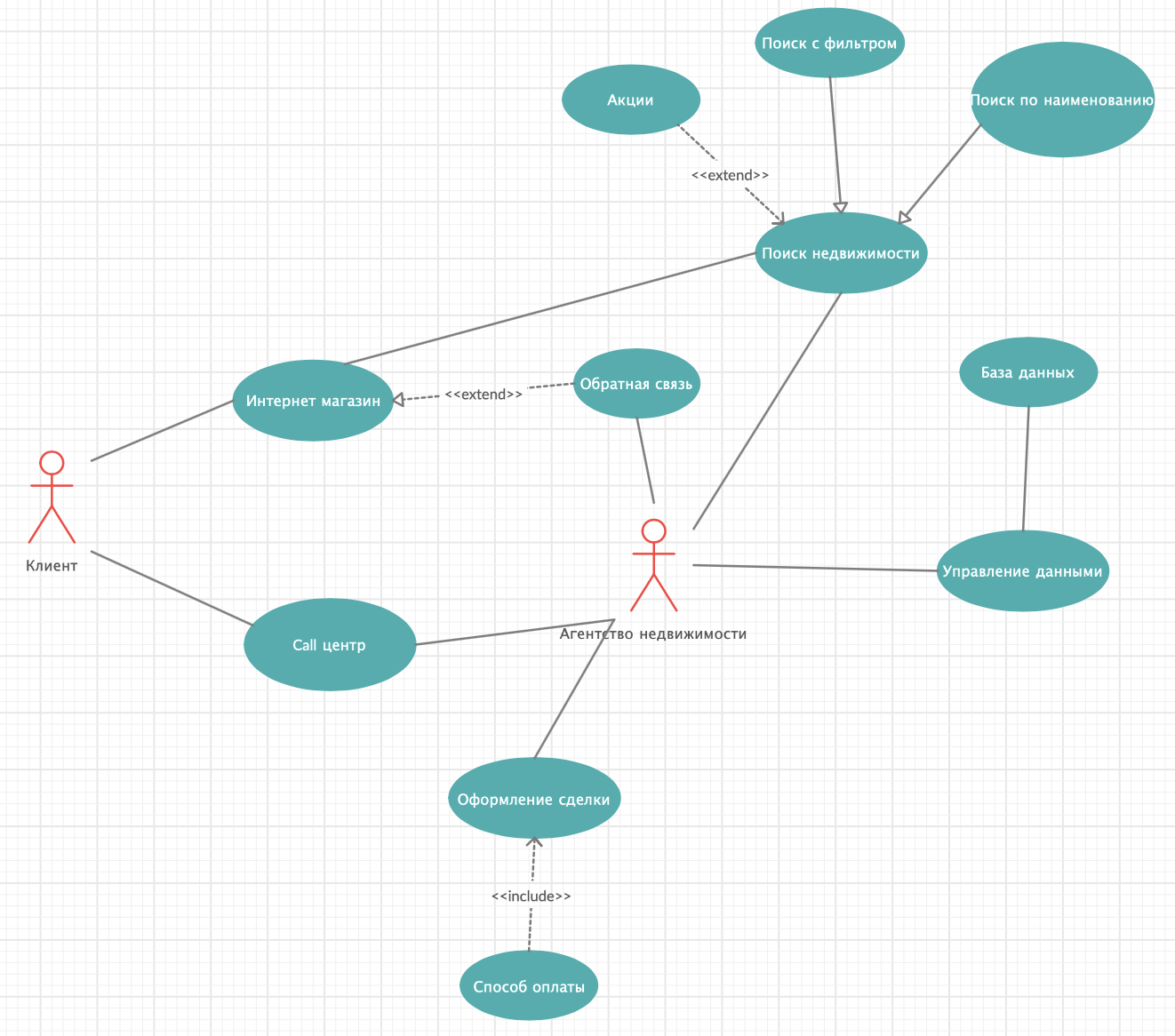
Разрабатываемая информационная система предназначена для осуществления работы основных процессов внутри магазина и его взаимодействия с внешними системами.

Система выполняет такие функции, как:

1. выбор недвижимости и оплата клиентом;
2. просмотр недвижимости;
3. просмотр недвижимости с применением различных фильтром поиска, акций и поиск по наименованию позиции в БД;
4. заключение договора между агентством недвижимости и клиентом;
5. быстрей online ответ клиенты через call-центр;

#### *Описание вариантов использования*

В диаграмме вариантов использования (рисунок 1) отражено функционирование информационной системы «Агентство недвижимости».



1. Диаграмма вариантов использования

Сценарий «Call центр» позволяет пользователю позвонить в агентство недвижимости.

Сценарий «Интернет магазин» позволяет Клиенту зайти на сайт в интернете по его адресу.

Сценарий «Обратная связь» позволяет Клиенту заказать обратный звонок или получить ответ на свой вопрос на почту.

Сценарий «Оформление сделки» позволяет Агентству недвижимости оформить сделку для Клиента.

Сценарий «Способ оплаты» позволяет выбрать удобный для Клиента способ оплаты недвижимости при оформлении сделки.

Сценарий «Управление данными» позволяет Агентству недвижимости добавлять/удалять/изменять записи в Базе данных.

Сценарий «База данных» предоставляет возможность хранить в себе данные о недвижимости.

Сценарий «Поиск недвижимости» позволяет Клиенту и Агентству недвижимости искать недвижимость.

Сценарий «Акции» позволяет искать недвижимость со скидкой.

Сценарий «Поиск с фильтром» позволяет искать недвижимость по заданным критериям.

Сценарий «Поиск по наименованию» позволяет пользователю искать недвижимость по наименованию, предусмотреть поиск по части слова.

#### *Расчет количественной оценки информационной наполненности диаграммы вариантов использования*

Для оценки диаграмм UML используется методика, позволяющая рассчитать количественный коэффициент, описывающий такие параметры диаграммы как наглядность, удобство восприятия, степень информативности (или загруженности информацией). Для расчета данного коэффициента используется формула:

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

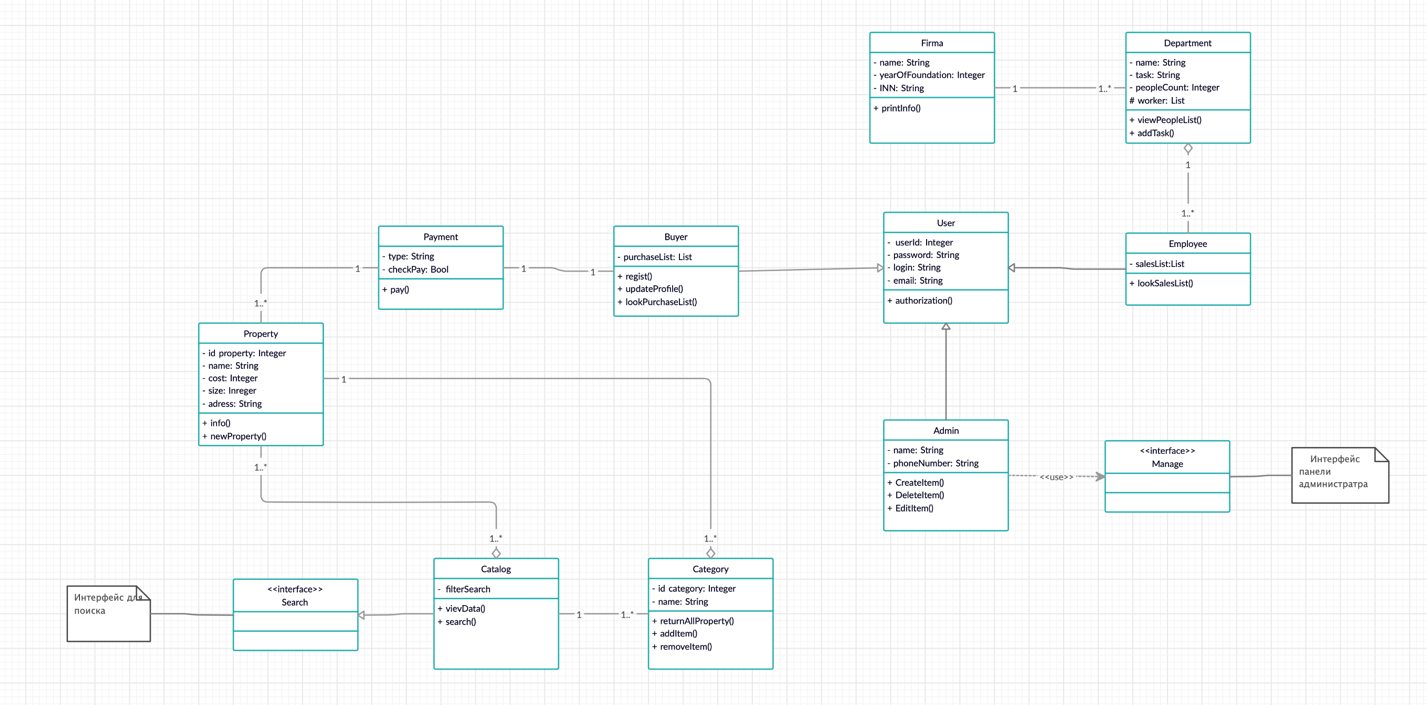
Для данной диаграммы оценка вычислена по формуле 1 и равна:

### 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 и типы отношений.

#### *Описание диаграммы классов*

Диаграмма классов (рисунок 2) служит для наглядного взаимодействия классов системы.



1. Диаграмма классов

#### *Расчет количественной оценки информационной наполненности диаграммы классов*

Расчёт количественной оценки информационной наполненности диаграммы классов оценивается по формулам 1-2.

Оценки классов:

Оценка диаграммы:

Эта оценка незначительно превышает оптимальный диапазон для этого вида диаграмм (5-5,5), но за счёт этого мы получаем более полное представление о взаимодействии классов в системе.

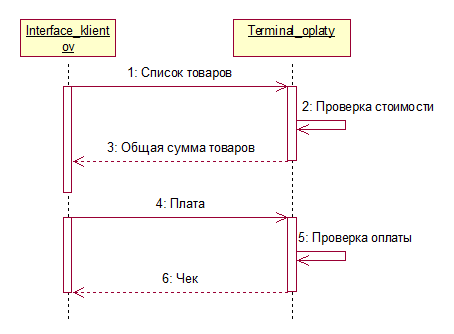
### Диаграммы взаимодействия

*Диаграмма последовательности* – это диаграмма, отображающая взаимодействия между объектами в рамках текущего сценария, упорядоченные во времени.

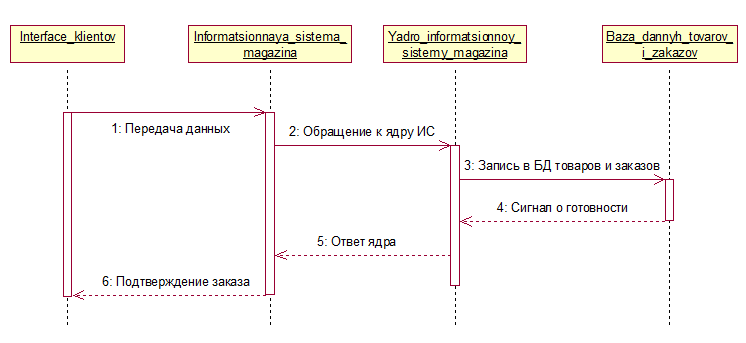
#### *Описания диаграмм последовательности*



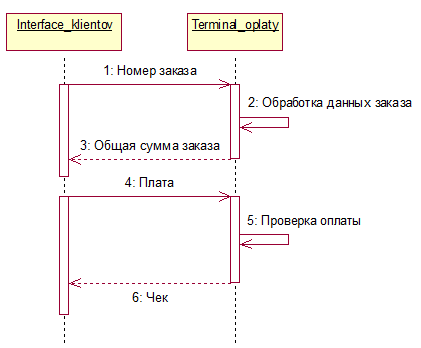
1. Диаграмма последовательности «Выбор товара»



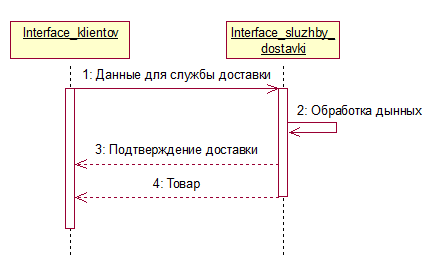
1. Диаграмма последовательности «Оплата товара»



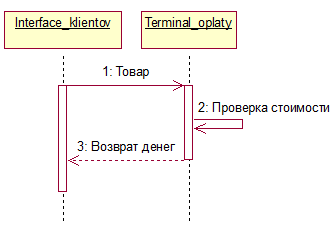
1. Диаграмма последовательности «Оформление заказа»



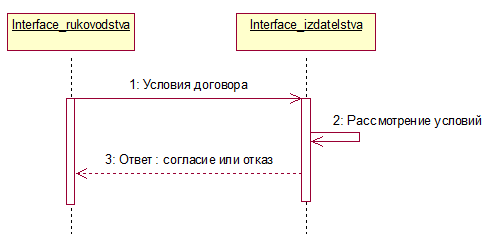
1. Диаграмма последовательности «Оплата заказа»



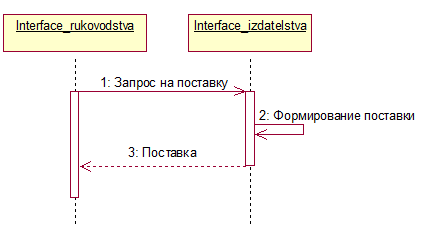
1. Диаграмма последовательности «Доставка товара»



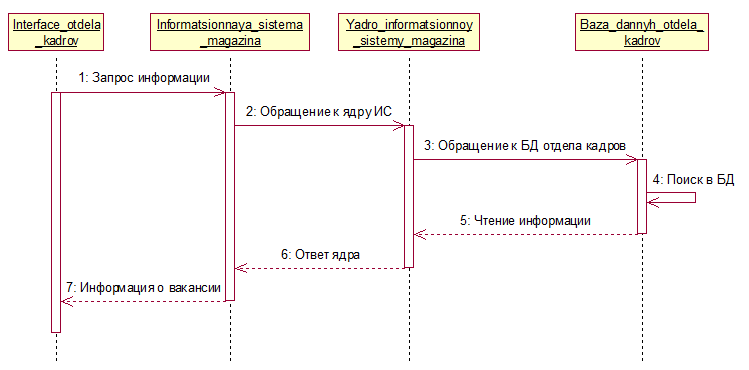
1. Диаграмма последовательности «Возврат товара»



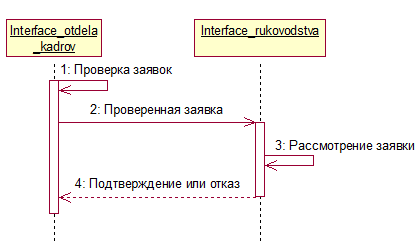
1. Диаграмма последовательности «Заключение договора с издательством»



1. Диаграмма последовательности «Поставка товара»



1. Диаграмма последовательности «Просмотр вакансий»



1. Диаграмма последовательности «Приём на работу»

#### *Расчет количественной оценки информационной наполненности диаграмм последовательности*

Расчёт количественной оценки информационной наполненности диаграмм последовательности оценивается по формуле 1 и представлен в таблице 4.

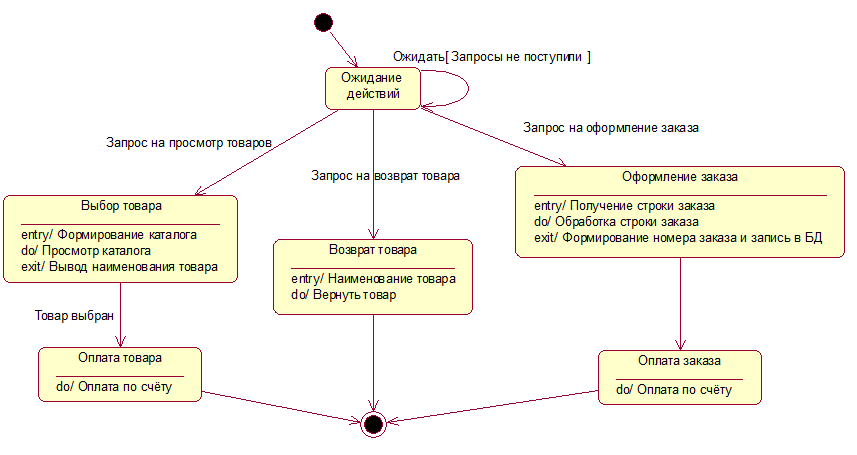
1. Оценка диаграмм последовательности

|  |  |
| --- | --- |
| Диаграмма последовательности | Оценка |
| Выбор товара | 3.85 |
| Оплата товара | 3.38 |
| Оформление заказа | 3.86 |
| Оплата заказа | 3.38 |
| Доставка товара | 2.8 |
| Возврат товара | 2.6 |
| Заключение договора с издательством | 2.6 |
| Поставка товара | 2.6 |
| Просмотр вакансий | 3.85 |
| Приём на работу | 2.8 |

### Диаграммы состояний

*Диаграммы состояний* – это один из пяти видов диаграмм UML, предназначенных для моделирования динамических аспектов поведения систем. Диаграмма состояний показывает конечный автомат. И диаграммы деятельности, и диаграммы состояний подходят для моделирования жизненного цикла объекта. Однако в то время, как диаграмма видов деятельности демонстрирует поток управления от одной деятельности к другой через множество объектов, диаграмма состояний отображает поток управления от состояния к состоянию внутри отдельного объекта.

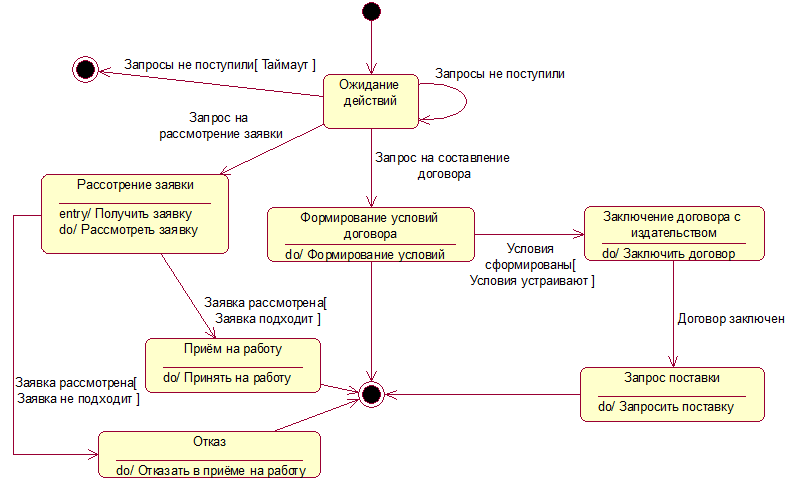
#### *Описание диаграмм состояний*



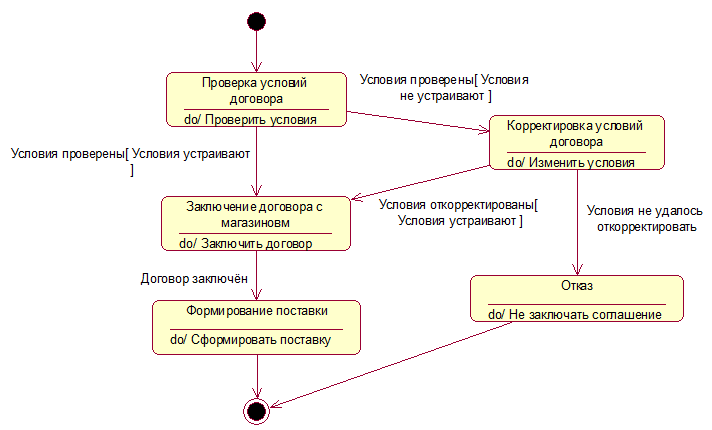
1. Диаграмма состояния «Interface\_klientov»



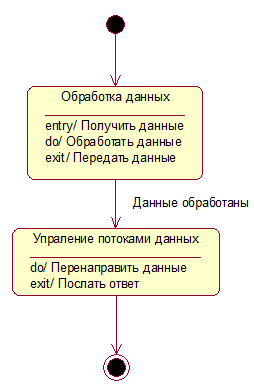
1. Диаграмма состояния «Terminal\_oplaty»



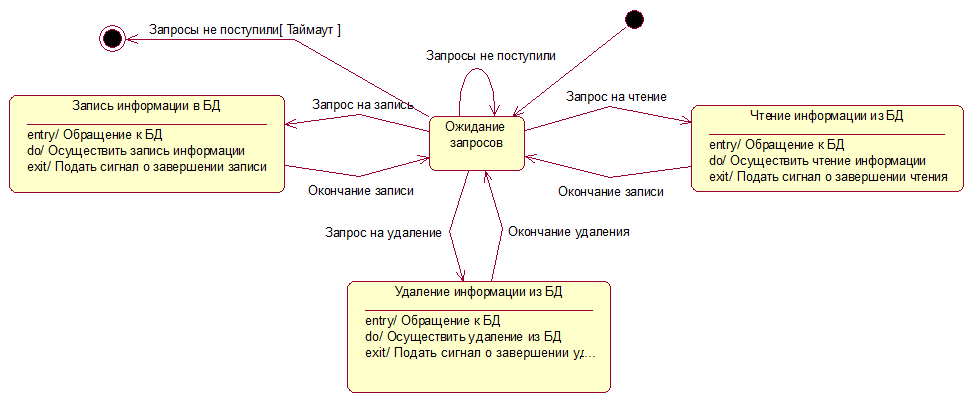
1. Диаграмма состояния «Interface\_rukovodstva»



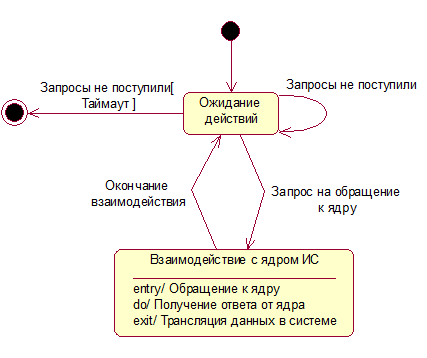
1. Диаграмма состояния «Interface\_izdatelstva»



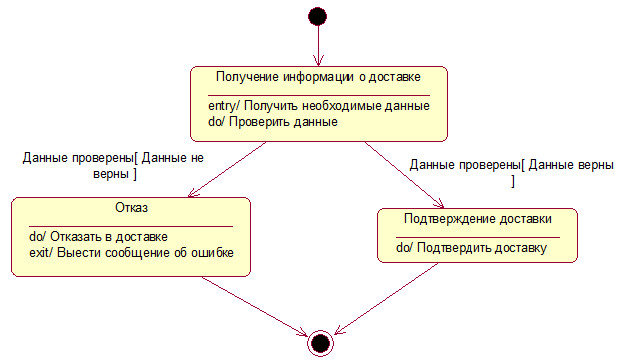
1. Диаграмма состояния «Yadro\_informatsionnoy\_sistemy\_magazina»



1. Диаграмма состояния «Baza\_dannyh\_otdela\_kadrov»



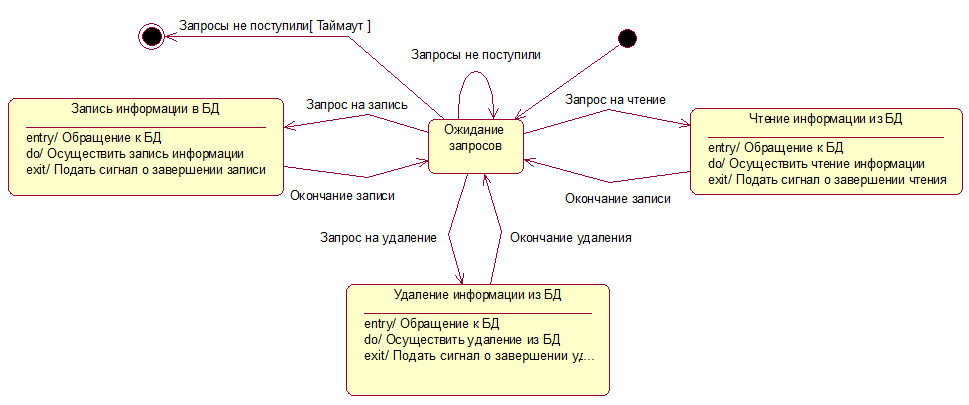
1. Диаграмма состояния «Informatsionnaya\_sistema\_magazina»



1. Диаграмма состояния «Interface\_sluzhby\_dostavki»



1. Диаграмма состояния «Interface\_otdela\_kadrov»



1. Диаграмма состояния «Baza\_dannyh\_tovarov\_i\_zakazov»

#### *Расчет количественной оценки информационной наполненности диаграмм состояний*

Расчёт количественной оценки информационной наполненности диаграмм состояний оценивается по формуле 1 и представлен в таблице 6.

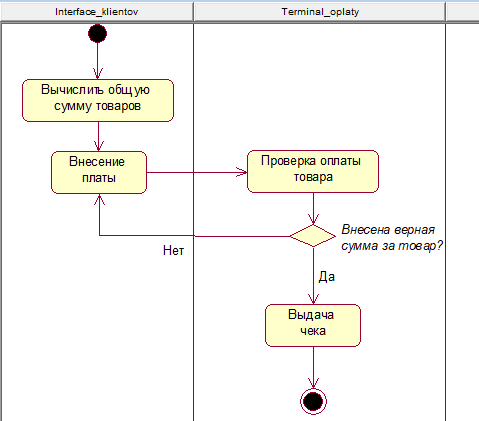
1. Оценка диаграмм состояний

|  |  |
| --- | --- |
| Диаграмма | Оценка |
| Interface\_klientov | 4.15 |
| Terminal\_oplaty | 3.74 |
| Interface\_rukovodstva | 4.35 |
| Interface\_izdatelstva | 3.64 |
| Yadro\_informatsionnoy\_sistemy\_magazina | 2.49 |
| Baza\_dannyh\_otdela\_kadrov | 3.89 |
| Informatsionnaya\_sistema\_magazina | 2.95 |
| Interface\_sluzhby\_dostavki | 3.14 |
| Interface\_otdela\_kadrov | 4.05 |
| Baza\_dannyh\_tovarov\_i\_zakazov | 3.89 |

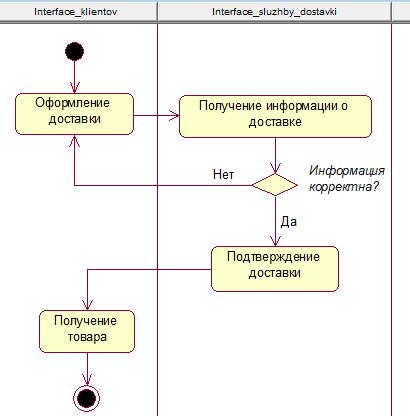
### Диаграммы видов деятельности

*Диаграммы видов деятельности* – это один из пяти видов диаграмм, применяемых в UML для моделирования динамических аспектов систем. По сути, диаграмма деятельности представляет собой блок-схему, которая показывает, как поток управления переходит от одной деятельности к другой. В отличие от традиционной блок-схемы диаграмма деятельности показывает параллелизм так же хорошо, как и ветвление потока управления.

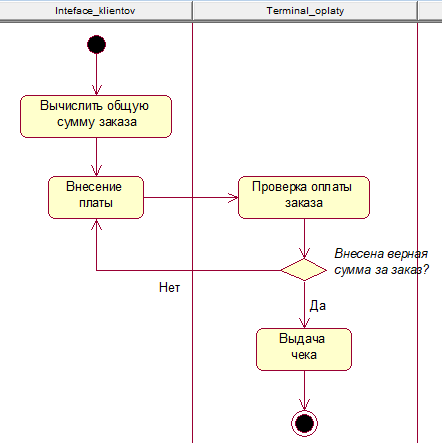
#### *Описания диаграмм видов деятельности*



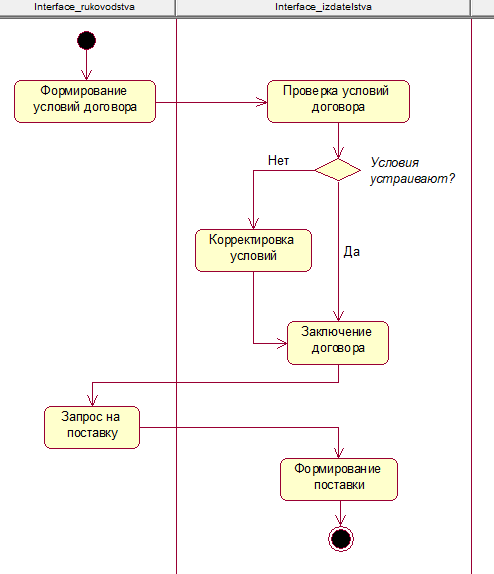
1. Диаграмма видов деятельности «Оплата товара»



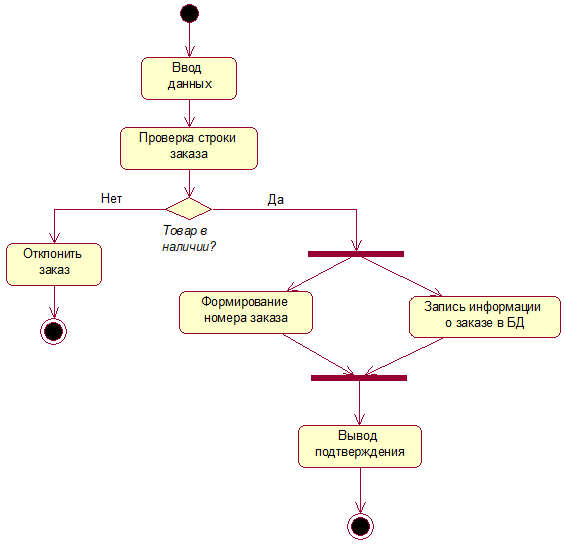
1. Диаграмма видов деятельности «Доставка товара»



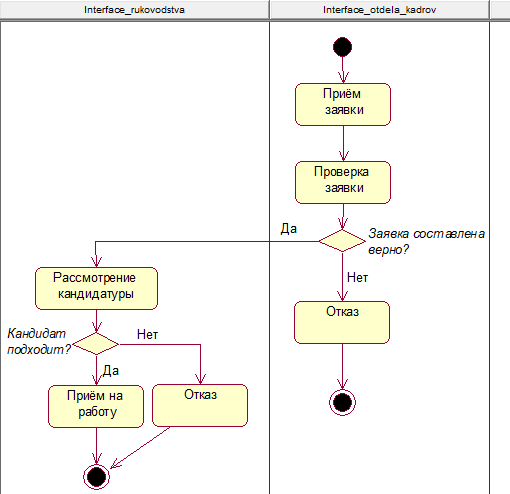
1. Диаграмма видов деятельности «Оплата заказа»



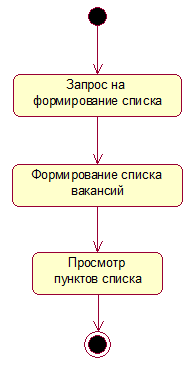
1. Диаграмма видов деятельности «Заключение договора на поставку»



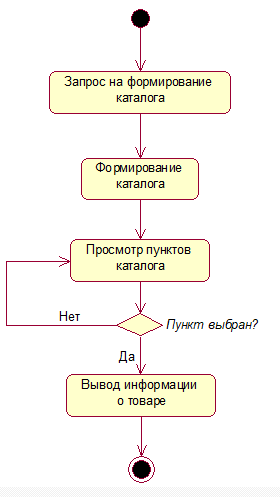
1. Диаграмма видов деятельности «Оформление заказа»



1. Диаграмма видов деятельности «Приём на работу»



1. Диаграмма видов деятельности «Просмотр вакансий»



1. Диаграмма видов деятельности «Выбор товара»

#### *Расчет количественной оценки информационной наполненности диаграмм видов деятельности*

Расчёт количественной оценки информационной наполненности диаграмм видов деятельности оценивается по формуле 1 и представлен в таблице 7.

1. Оценка диаграмм видов деятельности

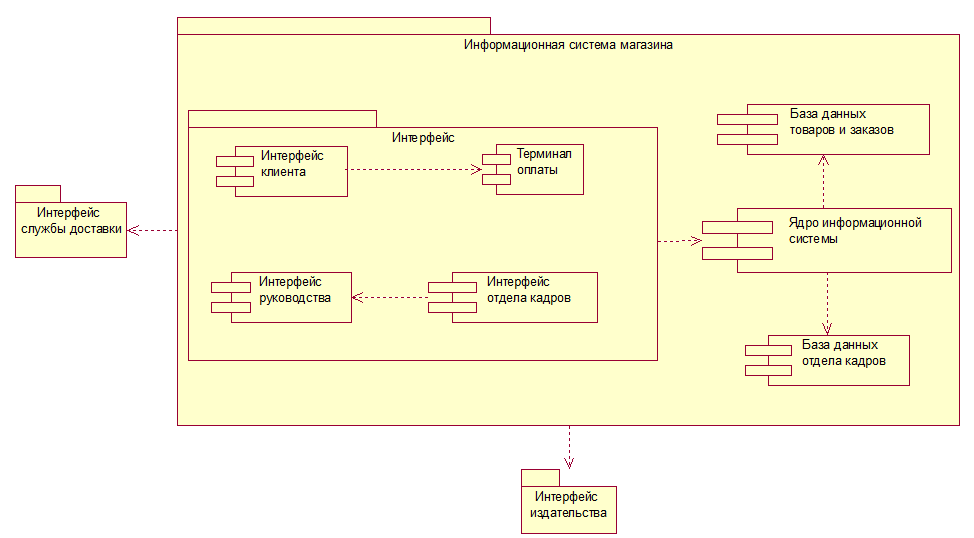
|  |  |
| --- | --- |
| Диаграмма последовательности | Оценка |
| Оплата товара | 2.97 |
| Доставка товара | 2.97 |
| Оплата заказа | 2.97 |
| Заключение договора на поставку | 3.39 |
| Оформление заказа | 3 |
| Приём на работу | 3.26 |
| Просмотр вакансий | 2.96 |
| Выбор товара | 4.01 |

### Диаграмма пакетов

*Диаграмма пакетов* – это диаграмма, представляющая логическую группировку составляющих информационную систему процессов и объектов. Данная диаграмма предназначена для обеспечения удобства разработки, так как позволяет группировать объекты любым способов. При этом один объект или пакет может принадлежать нескольким родительским пакетам.

#### *Описание диаграммы пакетов*

Диаграмма пакетов (рисунок 41) показывает, как хранятся, располагаются и взаимодействую между собой объекты информационной системы.



1. Диаграмма пакетов

#### *Расчет количественной оценки информационной наполненности диаграммы пакетов*

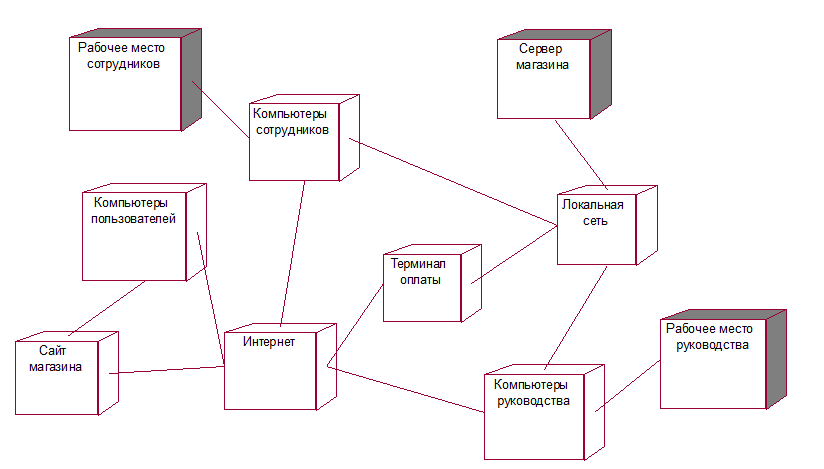
Расчёт количественной оценки информационной наполненности диаграмм пакетов оценивается по формуле 1.

### Диаграмма размещения

*Диаграмма размещения* – это диаграмма, предназначенная для моделирования работающих узлов (аппаратных средств), и компонентов информационной системы, развернутых на них.

#### *Описание диаграммы размещения*

Диаграмма развёртывания представлена на рисунке 42.



1. Диаграмма размещения

#### *Расчет количественной оценки информационной наполненности диаграммы размещения*

Расчёт количественной оценки информационной наполненности диаграмм размещения оценивается по формуле 1.

# ЗАКЛЮЧЕНИЕ

В данной курсовой работе была спроектирована информационная система «Книжный магазин» с помощью языка UML. Информационная система включает в себя: диаграмму вариантов использования (Use Case Diagram), диаграмму классов (Class Diagram), диаграммы последовательности (Sequence Diagram), диаграммы кооперации (Collaboration Diagram), диаграммы состояний (Statechart Diagram), диаграммы видов деятельности (Activity Diagram), диаграмму компонентов (Component Diagram) и диаграмму развертывания (Deployment Diagram).

Спроектированная система является наглядной основой для дальнейших модификаций, результатом которых станет возможность перехода к разработке полноценной информационной системы. На данном же этапе система упрощает работу с имеющимися данными и позволяет эффективно решать необходимые задачи. Rational Rose является универсальным средством моделирования на языке UML, которое позволяет быстро создать необходимую информационную систему и продуктивно ей пользоваться.

В результате генерации кода для каждого класс получены файлы с расширением .cpp и .h. Файлы с расширением .h содержат объявления функций, а файл с расширением .cpp содержат исходные коды классов.

# СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ

## Учебное пособие по КР ООПрИС 2020 Е.О. Пятлина, А.А.Ключарёв, Е.В. Павлов

## Книга: UML. Основы Фаулер М. 2020

## UML. Основы: краткое руководство по стандартному языку объектного моделирования 2016 Фаулер М.

## UML для простых смертных Роберт А. Максимчук 2016

## Введение в UML от создателей языка Гради Буч, Джеймс Рамбо, Ивар Якобсон 2015

# ПРИЛОЖЕНИE

Приложение А. Листинг сгенерированного кода

Admin.cpp

#include "Admin.h"

Admin::Admin(){}

initAttributes(){}

Admin::~Admin(){}

void Admin::initAttributes(){}

Admin.h

#ifndef ADMIN\_H

#define ADMIN\_H

#include "User.h"

#include <string>

class Admin : public User{

public:

Admin();

virtual ~Admin();

void createItem(){}

void DeleteItem(){}

void EditItem(){}

protected:

private:

std::string name;

std::string phoneNumber;

void setName(std::string new\_var){

name = new\_var;

}

std::string getName(){

return name;

}

void setPhoneNumber(std::string new\_var){

phoneNumber = new\_var;

}

std::string getPhoneNumber() {

return phoneNumber;

}

void initAttributes(){}

#endif // ADMIN\_H

Buyer.cpp

#include "Buyer.h"

Buyer::Buyer(){

initAttributes();

}

Buyer::~Buyer(){}

void Buyer::initAttributes(){}

Buyer.h

#ifndef BUYER\_H

#define BUYER\_H

#include "User.h"

#include <string>

#include <vector>

class Buyer : public User{

public:

Buyer();

virtual ~Buyer();

void regist(){}

void updateProfile(){}

void lookPurchaseList(){}

protected:

private:

std::string purchaseList;

void setPurchaseList(std::string new\_var){

purchaseList = new\_var;

}

std::string getPurchaseList(){

return purchaseList;

}

void initAttributes(){};

#endif // BUYER\_H

Catalog.cpp

#include "Catalog.h"

Catalog::Catalog(){

initAttributes();

}

Catalog::~Catalog(){}

void Catalog::initAttributes(){}

Category.h

#ifndef CATEGORY\_H

#define CATEGORY\_H

#include <string>

#include <vector>

class Category{

public:

Category();

virtual ~Category();

void returnAllProperty(){}

void addItem(){}

void removeItem(){}

protected:

private:

int idCategory;

std::string name;

void setIdCategory(int new\_var){

idCategory = new\_var;

}

int getIdCategory(){

return idCategory;

}

void setName(std::string new\_var){

name = new\_var;

}

std::string getName(){

return name;

}

void initAttributes();

};

#endif // CATEGORY\_H

Department.h

#ifndef DEPARTMENT\_H

#define DEPARTMENT\_H

#include <string>

#include <vector>

class Department{

public:

Department();

virtual ~Department();

void viewPeopleList(){}

void addTask(){}

protected:

std::string worker;

void setWorker(std::string new\_var) {

worker = new\_var;

}

std::string getWorker(){

return worker;

}

private:

std::string name;

std::string task;

int peopleCount;

void setName(std::string new\_var) {

name = new\_var;

}

std::string getName() {

return name;

}

void setTask(std::string new\_var){

task = new\_var;

}

std::string getTask() {

return task;

}

void setPeopleCount(int new\_var){

peopleCount = new\_var;

}

int getPeopleCount(){

return peopleCount;

}

void initAttributes();

};

#endif // DEPARTMENT\_H

Empluyee.h

#ifndef EMPLUYEE\_H

#define EMPLUYEE\_H

#include "User.h"

#include <string>

#include <vector>

class Empluyee : public User{

public:

Empluyee();

virtual ~Empluyee();

void lookSalesList(){}

protected:

private:

std::string salesList;

void setSalesList(std::string new\_var) {

salesList = new\_var;

}

std::string getSalesList(){

return salesList;

}

void initAttributes();

};

#endif // EMPLUYEE\_H

Firma.cpp

#include "Firma.h"

Firma::Firma(){

initAttributes();

}

Firma::~Firma(){}

void Firma::initAttributes(){}

Firma.h

#ifndef FIRMA\_H

#define FIRMA\_H

#include <string>

#include <vector>

class Firma{

public:

Firma();

virtual ~Firma();

void info() {}

protected:

private:

std::string name;

int yearOfFounder;

int INN;

void setName(std::string new\_var){

name = new\_var;

}

std::string getName(){

return name;

}

void setYearOfFounder(int new\_var){

yearOfFounder = new\_var;

}

int getYearOfFounder(){

return yearOfFounder;

}

void setINN(int new\_var) {

INN = new\_var;

}

int getINN(){

return INN;

}

void initAttributes();

};

#endif // FIRMA\_H

Payment.cpp

#include "Payment.h"

Payment::Payment(){

initAttributes();

}

Payment::~Payment(){}

void Payment::initAttributes(){}

Payment.h

#ifndef PAYMENT\_H

#define PAYMENT\_H

#include <string>

#include <vector>

class Payment{

public:

Payment();

virtual ~Payment();

void pay() {}

protected:

private:

std::string type;

bool checkPay;

void setType(std::string new\_var) {

type = new\_var;

}

std::string getType() {

return type;

}

void setCheckPay(bool new\_var) {

checkPay = new\_var;

}

bool getCheckPay(){

return checkPay;

}

void initAttributes();

};

#endif // PAYMENT\_H

Property.cpp

#include "Property.h"

Property::Property(){

initAttributes();

}

Property::~Property(){}

void Property::initAttributes(){}

Property.h

#ifndef PROPERTY\_H

#define PROPERTY\_H

#include <string>

#include <vector>

class Property{

public:

Property();

virtual ~Property();

void info(){}

void newProperty(){}

protected:

private:

int idProperty;

std::string name;

int cost;

int size;

std::string adress;

void setIdProperty(int new\_var) {

idProperty = new\_var;

}

int getIdProperty(){

return idProperty;

}

void setName(std::string new\_var){

name = new\_var;

}

std::string getName(){

return name;

}

void setCost(int new\_var) {

cost = new\_var;

}

int getCost() {

return cost;

}

void setSize(int new\_var) {

size = new\_var;

}

int getSize() {

return size;

}

void setAdress(std::string new\_var){

adress = new\_var;

}

std::string getAdress(){

return adress;

}

void initAttributes();

};

#endif // PROPERTY\_H

User.cpp

#include "User.h"

User::User(){

initAttributes();

}

User::~User(){}

void User::initAttributes(){}

User.h

#ifndef USER\_H

#define USER\_H

#include <string>

class User{

public:

User();

virtual ~User();

void authorization(){

protected:

private:

int userId;

std::string password;

std::string loign;

std::string email;

void setUserId(int new\_var){

userId = new\_var;

}

int getUserId(){

return userId;

}

void setPassword(std::string new\_var){

password = new\_var;

}

std::string getPassword(){

return password;

}

void setLoign(std::string new\_var){

loign = new\_var;

}

std::string getLoign() {

return loign;

}

void setEmail(std::string new\_var){

email = new\_var;

}

std::string getEmail() {

return email;

}

void initAttributes();

};

#endif // USER\_H