Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

Высшего профессионального образования

«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет электроники и вычислительной техники

Кафедра ЭВМиС

Семестровая работа

По дисциплине «Введение в разработку программного обеспечения»

Выполнил: студент

Группы ИВТ-460

Юров Д.И.

Проверил:

Андреев А.А.

Волгоград, 2015

Оглавление

[1 Описание объекта разработки 3](#_Toc408905109)

[2 Пользовательские истории 4](#_Toc408905110)

[3 Прецеденты 5](#_Toc408905111)

[4 Требования к аппаратному и программному обеспечению 7](#_Toc408905112)

[5 Модель классов предметной области 8](#_Toc408905113)

[7 План разработки 8](#_Toc408905114)

[8 UML диаграммы 10](#_Toc408905115)

[9 Паттерны 11](#_Toc408905116)

[10 Рефакторинг 13](#_Toc408905117)

[11 Внешний вид экранных форм 14](#_Toc408905118)

[12 Упаковка 18](#_Toc408905120)

[Приложение А – Исходный код программы 20](#_Toc408905121)

# Описание объекта разработки

Введение

В качестве предметной области данной работы была выбрана работа провайдера услуг интернета.

Основными элементами предметной области должны быть: пользователи, заключающие договора, денежные операции, которые они проводят совместно с провайдером услуг интернета, сотрудники, оформляющие договора, списки тарифных планов для каждого подключающегося клиента, а также данные о зоне покрытия для подключающихся клиентов.

Позиционирование

Система позиционируется как система с закрытой архитектурой, что позволяет автору удержать за собой ряд прав. Исходный код будет недоступен для пользователя, а также и модификация программы под определенные нужды.

Экономические предпосылки

Система позволит снизить сложность работы с большим количеством разнородных данных. Также не требуется отдельно выделенных рабочих мест и дополнительного штата операторов для ввода данных. Каждый сотрудник каждого отдела вводит в базу данных (БД) только «свою» информацию со своего рабочего места.

Формулировка проблемы

Проблема подобных систем - отсутствие доверия к ним при закрытости исходного кода. Но это уменьшает факт наличия человеческого фактора, из-за которого и возникает большинство ошибок.

Заинтересованные лица

Сотрудники провайдера услуг интернета хотят иметь удобное программное средство, которое полностью или частично избавило бы их от необходимости работы с договорами и иной информации фирмы вручную.

Перспективы

Есть.

Преимущества

Закрытая архитектура обеспечивает сохранность кода от несанкционированного изменения пользователем.

# Пользовательские истории

1. Составление договора или его изменение

Сотрудник может составлять договора с пользователями, которые присутствую в БД, заполнив необходимые поля с данными по текущему договору и нажав кнопку «Заключить», также сотрудник может изменить данные договора или расторгнуть договор, по нажатию соответствующей кнопки.

2. Добавление и удаление данных о зоне покрытия

Сотрудник может добавлять данные о зоне покрытия или вносить в них изменения. Сохранение изменений фиксируется нажатием кнопки «Добавить» или «Редактировать» в зависимости от действий.

3. Добавление, изменение, удаление нового пользователя

Сотрудник может внести данные нового пользователя в соответствующую форму «Пользователи», заполнив необходимые поля и нажав кнопку «Добавить». Здесь же можно просмотреть полный список всех пользователей в базе, также сотрудник может изменить данные пользователя или удалить запись с пользователем.

4. Добавление, изменение, удаление сотрудника

Сотрудник может добавить принятого на работу нового сотрудника в соответствующую форму «Сотрудники», заполнив необходимые поля с личной информацией о специалисте и нажав кнопку «Добавить». Здесь же в таблице можно просмотреть уже имеющихся в базе специалистов, также сотрудник может изменить данные сотрудника или удалить запись с сотрудником.

5. Добавление, изменение, удаление тарифного плана

Сотрудник может добавить новый тарифный план в соответствующую форму «Тарифные планы», заполнив необходимые поля с информацией о ТП и нажав кнопку «Добавить». Здесь же в таблице можно просмотреть уже действующие тарифные планы, также сотрудник может изменить данные ТП или удалить запись с ТП.

6. Выполнение денежной операции по выбранному договору

Сотрудник может выполнить операцию со счетом выбранного тарифного плана нажав по кнопке «Работа со счетом» формы со списком договоров, затем внеся данные о транзакции, выполнить операцию нажав на соответствующую кнопку.

# Прецеденты

П1 Добавление данных в Электронную карту

Основной исполнитель: Сотрудник.

Заинтересованные лица и их требования:

- Сотрудник – лицо, имеющее доступ к Электронной карте. Может вносить изменения в базу данных, добавлять новые данные о пациентах, редактировать записи, обращаться к базе в любое время, не обременяя себя на заполнение бумажной версии карты.

Предусловия:

Сотрудник запускает программу и может вносить изменения в карту пациента.

Постусловия:

Сохранение информации в базе данных.

Основной успешный сценарий:

1. Запускается пользовательское ПО.
2. Сотрудник выбирает форму GUI «Пользователи» для внесения данных.
3. Вводит в соответствующие поля данные нового пользователя и нажимает кнопку добавить
4. Затем сотрудник переходит на форму «Работа с договорами»
5. Потом сотрудник выбирает добавленного пользователя и вносит остальные данные необходимые для заключения договора, включая сотрудника оформившего договор и оговоренный с пользователем тарифный план.
6. Сотрудник добавляет новый договор в базу данных провайдера, нажав кнопку «Заключить»

Расширения:

\*а При непредвиденных сбоях и невозможности продолжить работу программы выдаёт сообщение об ошибке и немедленно закрывается.

1а Если данные некорректны, система сообщит о проблеме.

Диаграмма прецедентов

Полная диаграмма всех прецедентов изображена на Рисунке 1.

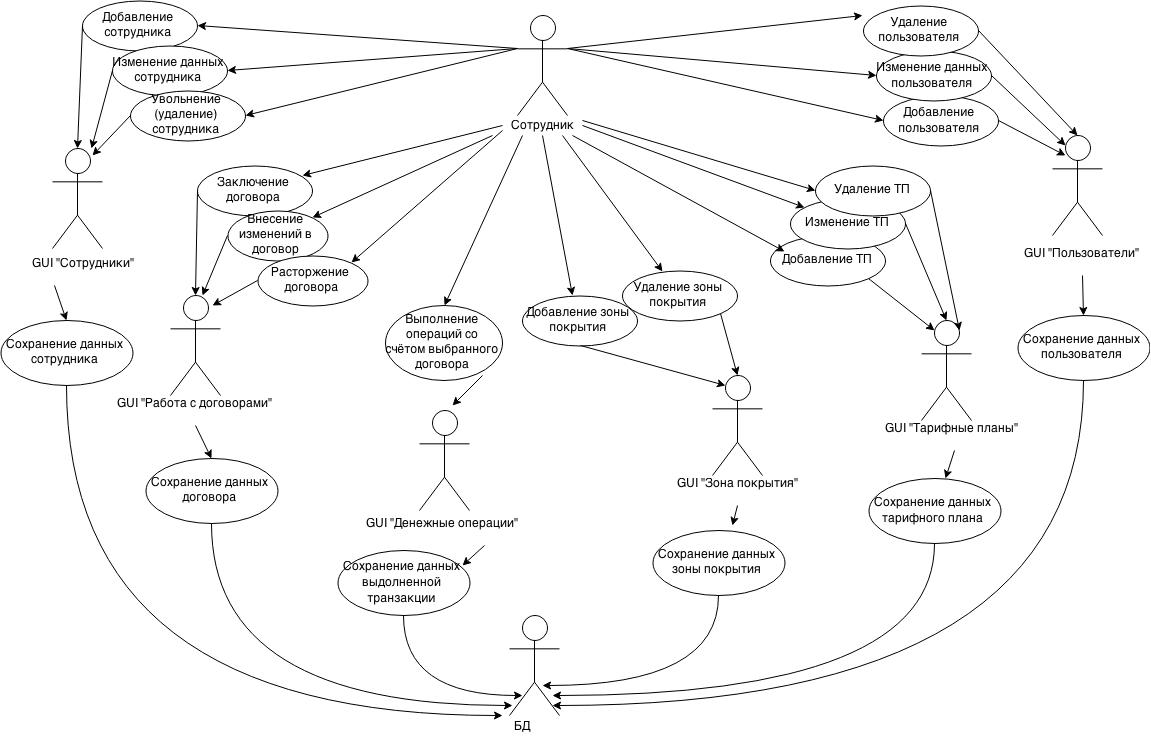


Рисунок 1 – Диаграмма прецедентов

# Требования к аппаратному и программному обеспечению

Операционная система: Microsoft Windows 7 или выше

Сервер базы данных: Microsoft SQL Server Express 2008 R2 или выше

Среда исполнения: Microsoft .NET Framework 4.5

Используемый язык: С#

Дополнительные технологии: LINQ TO SQL

# Модель классов предметной области

Кандидаты на роль концептуальных классов:

1. Договор

2. Сотрудник

3. Пользователь

4. Тарифный план

5. Зона покрытия

6. Денежная операция

# 7 План разработки

Подробный план разработки представлен в Таблицах 1, 2, 3.

Таблица 1. План выпуска версий системы.

|  |  |  |  |
| --- | --- | --- | --- |
| Пожелания Клиента | Оценка трудозатрат | Реализовать в итерации № | Реализовать в версии № |
| Создание базы данных | 1 | 1 | 1 |
| Формирование результатов | 1 | 1 | 1 |
| Подключение авторизации | 1 | 2 | 1 |
| Сохранение и редактирование данных через графический интерфейс | 1 | 2 | 1 |
| Основной GUI представления иерархии данных | 1 | 2 | 1 |

Таблица 2. Подробный план первой итерации

| Задача | Оценка трудозатрат |
| --- | --- |
| Создание базы данных | 10 |
| Создание проекта и окружения | 1 |
| Первичное архитектурное проектирование | 1 |
| Создание модульных тестов | 2 |
| Разработка классов БД | 1 |
| Разработка сервисных классов | 1 |
| Создание минимально-необходимого GUI | 1 |
| Разработка формирования результатов | 1 |
| Планирование на следующую итерацию | 0.5 |
| Итого: | 8.5 |

Таблица 3. Подробный план второй итерации

|  |  |
| --- | --- |
| Задача | Оценка трудозатрат В ИД |
| Основной GUI представления иерархии данных | 10 |
| Создание главного меню | 1 |
| Создание формы редактирования отдельных элементов данных | 1 |
| Разработка экспорта содержимого | 1 |
| Дополнение сервисных классов | 0.5 |
| Создание логики управления содержимым | 2 |
| Создание модульных тестов | 2 |
| Итого: | 7.5 |

# 8 UML диаграммы

На Рисунке 2 представлена диаграмма классов

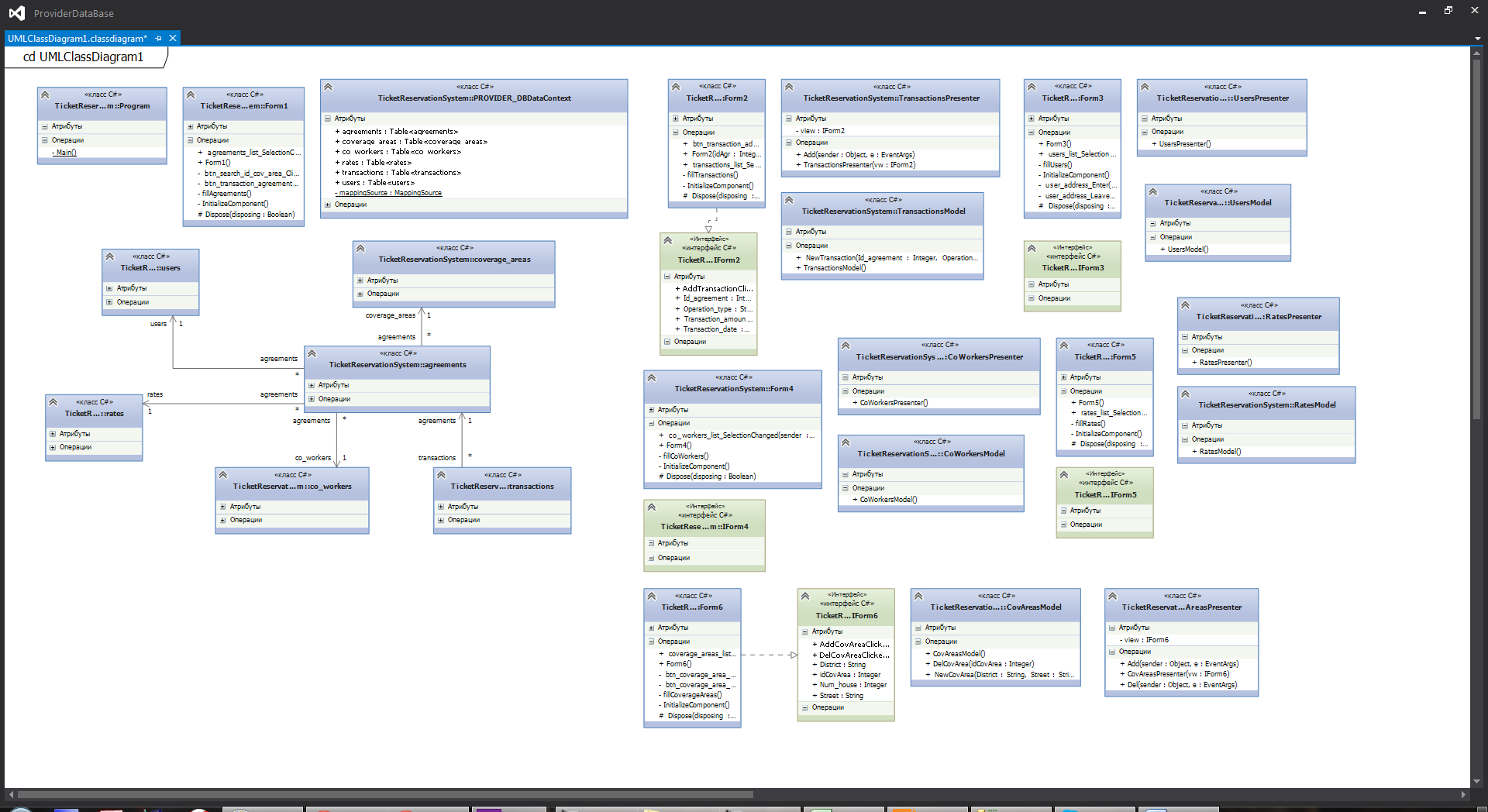


Рисунок 2 – Диаграмма классов после написания программы

Среда разработки Visual Studio, начиная с версии 2008 года, поддерживает автоматическое создание диаграмм классов. (Однако данное построение доступно только в редакции Professional).  
  
Рассмотрим процесс создание диаграммы на примере среды Visual Studio 2012.  
  
1. Для начала необходимо создать сам файл с будущей диаграммой.

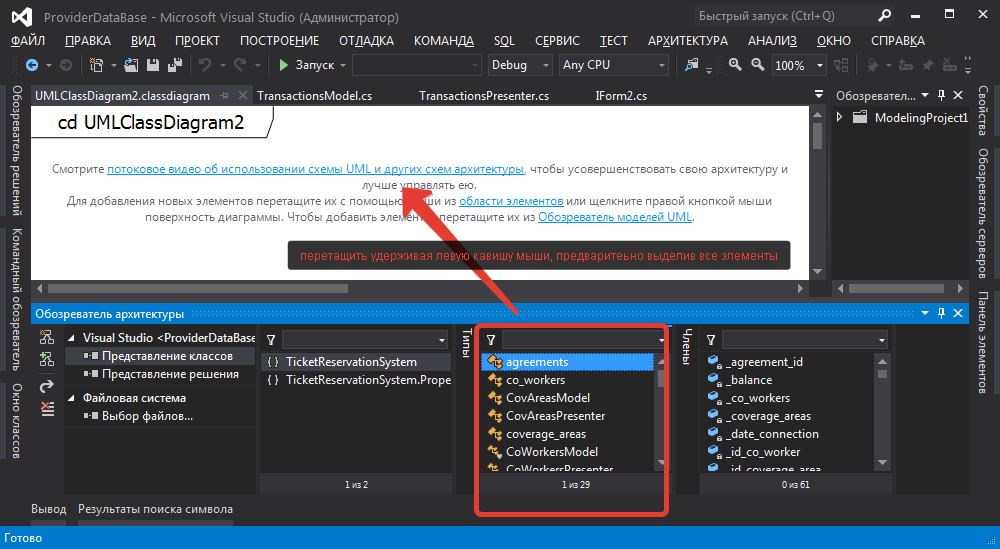
АРХИТЕКТУРА 🡪 Создать схему 🡪 Схема классов UML 🡪 Ок

2. Затем открыть Обозреватель архитектуры

АРХИТЕКТУРА 🡪 Окна 🡪 Обозреватель архитектуры

3. В обозревателе архитектуры перейти выбрать класс (пространство имён) своего приложения

Обозреватель архитектуры 🡪 Представления классов 🡪 [основной класс проекта]

4. Перетащить все элементы из вкладки Типы на область моделей UML

5. Выровнять при необходимости построенные диаграммы

Диаграмма пакетов представлена на Рисунке 3.

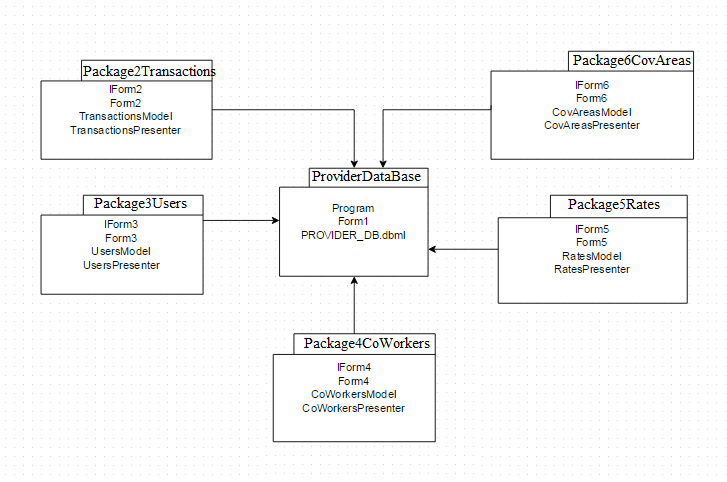


Рисунок 3 – Диаграмма пакетов.

# Паттерны

В данной работе можно продемонстрировать следующие паттерны:

1. Model-View-Presenter – архитектурный паттерн, применяется для отделения представления от модели данных, в соответствии с ролями в этом паттерне была выполнена упаковка проекта;
2. Наблюдатель – применялся для реакции на действия пользователя – окна по определенным событиям сообщали Presenter’ам о пользовательских действиях.

Данные паттерны можно увидеть на диаграмме классов. Для каждой формы, они же классы View (кроме первой) был выделен интерфейс методов (IForm), классы Presenter’ов и модели, которые обрабатывали пользовательское действия. По нажатию кнопки событие в классе формы посылает сигнал Presenter’у, чья форма была задействована. Presenter определяет событие какой кнопки нужно обработать и вызывает соответствующую функцию модели (Model), которая обрабатывает пользовательское действие.

Пример:

Возьмем за пример обработку формы «Сотрудники»

Интерфейс:

namespace TicketReservationSystem

{

public interface IForm2

{

int Id\_agreement { get; }

string Operation\_type { get; }

int Transaction\_amount { get; }

DateTime Transaction\_date { get; }

event EventHandler<EventArgs>AddTransactionClicked;

}

}

Использование события для перехвата пользовательского действия (добавление сотрудника):

// ДОБАВЛЕНИЕ НОВОЙ ТРАНЗАКЦИИ

public void btn\_transaction\_add\_Click(object sender, EventArgs e)

{

if (operation\_type.Text != "" && transaction\_amount.Value != 0)

{

// НАБЛЮДАТЕЛЬ

AddTransactionClicked(this, e)

// обновим таблицу

fillTransactions();

}

else

{

MessageBox.Show("Не все поля заполнены верно!");

}

}

Класс Presentor’а, обрабатывающего данное событие и передающий параметры для Модели:

namespace TicketReservationSystem

{

public class TransactionsPresenter

{

IForm2 view;

public TransactionsPresenter(IForm2 vw)

{

view = vw;

view.AddTransactionClicked += new EventHandler<EventArgs>(Add);

}

public void Add(object sender, EventArgs e)

{

//Обращаемся к модели

TransactionsModel tranMod = new TransactionsModel();

tranMod.NewTransaction(view.Id\_agreement, view.Operation\_type,

view.Transaction\_amount, view.Transaction\_date);

}

}

}

Модель для обработки данных:

namespace TicketReservationSystem

{

public class TransactionsModel

{

public void NewTransaction(int Id\_agreement, string Operation\_type,

int Transaction\_amount, DateTime Transaction\_date)

{

PROVIDER\_DBDataContext dc = new PROVIDER\_DBDataContext();

var Trans = new transactions()

{

id\_agreement = Id\_agreement,

operation\_type = Operation\_type,

transaction\_amount = Transaction\_amount,

transaction\_date = Transaction\_date

};

dc.transactions.InsertOnSubmit(Trans); // добавление в БД

dc.SubmitChanges(); // выполняет соответствующие команды

// для внесения изменений базы данных

}

}

}

# Рефакторинг

Наиболее часто использовавшиеся рефакторинги:

1. Переименование переменной/метода – данный вид рефакторинга применялся для обеспечения большей наглядности кода, удаления лишних комментариев.

До применения рефакторинга:

namespace projeckt1

{

public partial class Form1 : Form

{

PROVIDER\_DBDataContext dc;

Form3 f3;

Form4 f4;

Form5 f5;

Form2 f2;

Form6 f6;

public Form1()

{

InitializeComponent();

f3 = new Form3();

f4 = new Form4();

f5 = new Form5();

f3.Show();

f4.Show();

f5.Show();

dc = new PROVIDER\_DBDataContext();

fillAgreements();

agreementOk.Visible = false;

agreementCancel.Visible = false;

comboBox1.Items.Clear();

comboBox2.Items.Clear();

comboBox3.Items.Clear();

comboBox4.Items.Clear();

var query1 = dc.users;

foreach (var item in query1) comboBox1.Items.Add(item.user\_name.ToString());

var query2 = dc.coverage\_areas;

foreach (var item in query2) comboBox2\_area.Items.Add(item.coverage\_area\_id.ToString());

var query3 = dc.co\_workers;

foreach (var item in query3) comboBox3.Items.Add(item.co\_worker\_name.ToString());

var query4 = dc.rates;

foreach (var item in query4) comboBox4.Items.Add(item.name\_rate.ToString());

if (comboBox1.Items.Count > 0) comboBox1.SelectedIndex = 0;

if (comboBox2\_area.Items.Count > 0) comboBox2\_area.SelectedIndex = 0;

if (comboBox3.Items.Count > 0) comboBox3.SelectedIndex = 0;

if (comboBox4.Items.Count > 0) comboBox4.SelectedIndex = 0;

// Перезаполним таблицу

int countRow = agreements\_list.Rows.Count;

for (int i = 0; i < countRow; i++)

{

dataGridView.Rows.RemoveAt(0);

}

var query5 = dc.agreements;

int numRow = 0;

foreach (var item in query5)

{

dataGridView.Rows.Add();

dataGridView Rows[numRow].Cells[0].Value = item.agreement\_id.ToString();

dataGridView Rows[numRow].Cells[1].Value = item.users.user\_name.ToString();

dataGridView Rows[numRow].Cells[2].Value = item.balance.ToString();

dataGridView Rows[numRow].Cells[3].Value = item.login.ToString();

dataGridView Rows[numRow].Cells[4].Value = item.password.ToString();

dataGridView Rows[numRow].Cells[5].Value = item.id\_coverage\_area.ToString();

dataGridView Rows[numRow].Cells[6].Value = item.num\_apartment.ToString();

dataGridView Rows[numRow].Cells[7].Value = item.date\_connection.ToString();

dataGridView Rows[numRow].Cells[8].Value = item.co\_workers.co\_worker\_name.ToString() + " " +

item.co\_workers.co\_worker\_surname.ToString();

dataGridView \_list.Rows[numRow].Cells[9].Value = item.rates.name\_rate.ToString();

numRow++;

}

После применения рефакторинга:

namespace TicketReservationSystem

{

public partial class Form1 : Form

{

PROVIDER\_DBDataContext dc;

Form3 f3;

Form4 f4;

Form5 f5;

Form2 f2;

Form6 f6;

public Form1()

{

InitializeComponent();

f3 = new Form3();

f4 = new Form4();

f5 = new Form5();

f3.Show();

f4.Show();

f5.Show();

dc = new PROVIDER\_DBDataContext();

fillAgreements();

agreementOk.Visible = false;

agreementCancel.Visible = false;

}

// Заполнение таблицы договоров

private void fillAgreements()

{

// Перезаполним ComboBox-ы

id\_user.Items.Clear();

id\_coverage\_area.Items.Clear();

id\_co\_worker.Items.Clear();

id\_rate.Items.Clear();

var query1 = dc.users;

foreach (var item in query1) id\_user.Items.Add(item.user\_name.ToString());

var query2 = dc.coverage\_areas;

foreach (var item in query2) id\_coverage\_area.Items.Add(item.coverage\_area\_id.ToString());

var query3 = dc.co\_workers;

foreach (var item in query3) id\_co\_worker.Items.Add(item.co\_worker\_name.ToString());

var query4 = dc.rates;

foreach (var item in query4) id\_rate.Items.Add(item.name\_rate.ToString());

if (id\_user.Items.Count > 0) id\_user.SelectedIndex = 0;

if (id\_coverage\_area.Items.Count > 0) id\_coverage\_area.SelectedIndex = 0;

if (id\_co\_worker.Items.Count > 0) id\_co\_worker.SelectedIndex = 0;

if (id\_rate.Items.Count > 0) id\_rate.SelectedIndex = 0;

// Перезаполним таблицу

int countRow = agreements\_list.Rows.Count;

for (int i = 0; i < countRow; i++)

{

agreements\_list.Rows.RemoveAt(0);

}

var query5 = dc.agreements;

int numRow = 0;

foreach (var item in query5)

{

agreements\_list.Rows.Add();

agreements\_list.Rows[numRow].Cells[0].Value = item.agreement\_id.ToString();

agreements\_list.Rows[numRow].Cells[1].Value = item.users.user\_name.ToString();

agreements\_list.Rows[numRow].Cells[2].Value = item.balance.ToString();

agreements\_list.Rows[numRow].Cells[3].Value = item.login.ToString();

agreements\_list.Rows[numRow].Cells[4].Value = item.password.ToString();

agreements\_list.Rows[numRow].Cells[5].Value = item.id\_coverage\_area.ToString();

agreements\_list.Rows[numRow].Cells[6].Value = item.num\_apartment.ToString();

agreements\_list.Rows[numRow].Cells[7].Value = item.date\_connection.ToString();

agreements\_list.Rows[numRow].Cells[8].Value = item.co\_workers.co\_worker\_name.ToString() + " " +

item.co\_workers.co\_worker\_surname.ToString();

agreements\_list.Rows[numRow].Cells[9].Value = item.rates.name\_rate.ToString();

numRow++;

}

}

}

}

1. Выделение метода – данный рефакторинг производился при разработке Presenter’ов для упрощения кода, уменьшения размера методов и более четкого разделения действий. Под каждый Presentor был выделен свой класс и интерфейс, что наглядно продемонстрировало работу программы и упростило код программы. Данное распределение можно увидеть на диаграмме пакетов (Рисунок 3).

# Внешний вид экранных форм

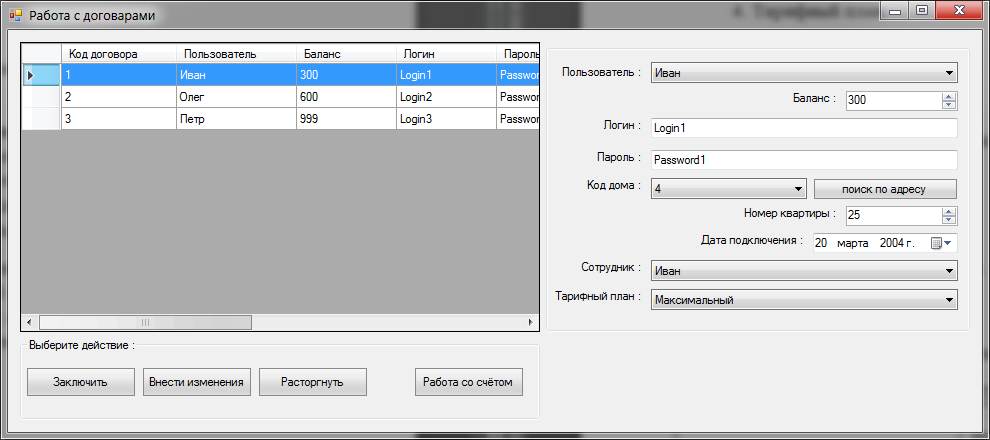


Рисунок 4 – Работа с договорами

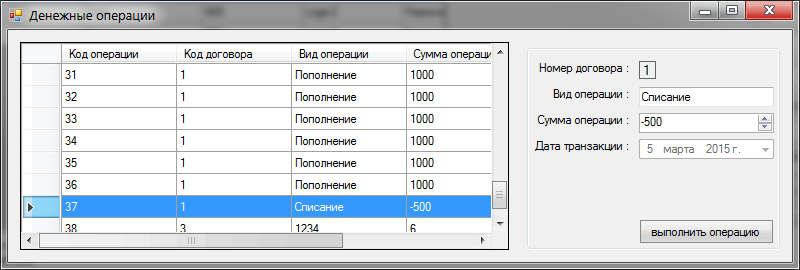


Рисунок 5 – Работа со счётом выбранного договора

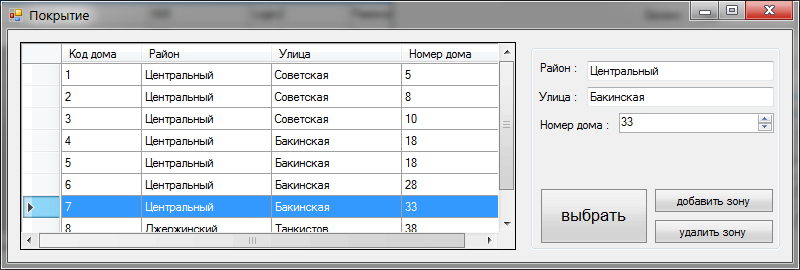


Рисунок 6 – Работа с зонами покрытия

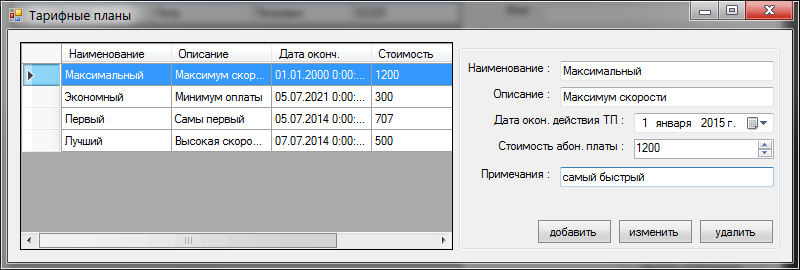


Рисунок 7 – Работа с тарифными планами

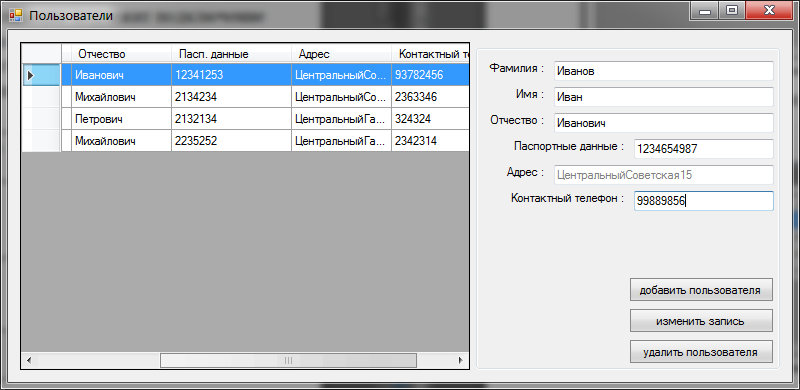


Рисунок 8 – Работа со списком пользователей

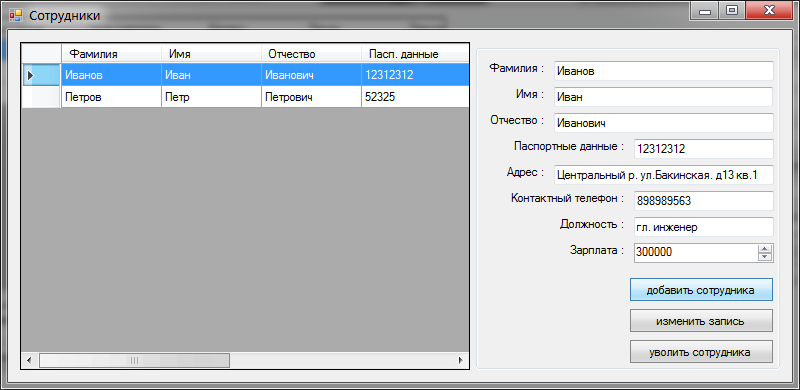


Рисунок 9 – Работа со списком сотрудников

# Упаковка

Обозначения метрик имеют следующий смысл:

N – Просто количество классов / модулей в пакете;

NA – Количество абстрактных классов (интерфейсов) в пакете;

Ca – Центростремительное связывание (сколько классов из других пакетов зависят от классов этого пакета);

Ce – Центробежное связывание (от скольких классов из других пакетов зависят классы этого пакета);

R – Количество связей внутри пакета;

H – Относительное сцепление, H = (R + 1) / N;

I – Неустойчивость, I = Ce / (Ce + Ca);

A – Абстрактность, А = NA / N;

D – Расстояние до главной последовательности, D = | A + I – 1| / 20,5 ;

D' – Нормализованное расстояние, D' = | A + I – 1|.

Расчет метрики будет произведен на основе диаграммы пакетов.

Метрики пакета Package2Transactions

N = 4 NA = 1 Ca = 3 Ce = 5 R = 4 H = 1.25 I = 0.62 A = 0.25

D = 0.09 D' = 0.13

Метрики пакета Package3Users:

N = 4 NA = 1 Ca = 5 Ce = 2 R = 4 H = 1.25 I = 0.29 A = 0.25

D = 0.33 D' = 0.46

Метрики пакета Package4CoWorkers:

N = 4 NA = 1 Ca = 3 Ce = 2 R = 4 H = 1.25 I = 0.4 A = 0.25

D = 0.25 D' = 0.35

Метрики пакета ProviderDataBase:

N = 4 NA = 0 Ca = 4 Ce = 14 R = 2 H = 0.75 I = 0.8 A = 0

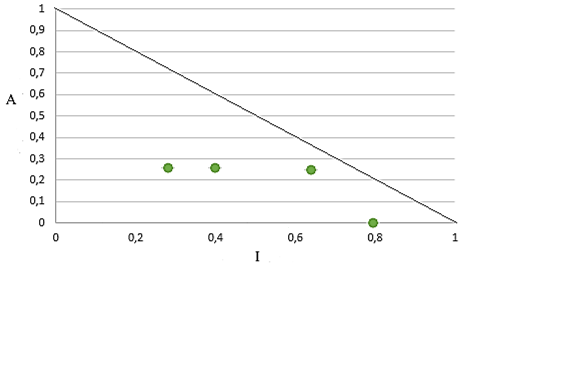
D = 0.14 D' = 0.2 

Рисунок 8 – Метрики

# Приложение А – Исходный код программы

Интерфейсы:

namespace TicketReservationSystem

{

public interface IForm2

{

int Id\_agreement { get; }

string Operation\_type { get; }

int Transaction\_amount { get; }

DateTime Transaction\_date { get; }

event EventHandler<EventArgs>AddTransactionClicked;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TicketReservationSystem

{

public interface IForm6

{

string District { get; }

string Street { get; }

int Num\_house { get; }

int idCovArea { get; }

event EventHandler<EventArgs> AddCovAreaClicked;

event EventHandler<EventArgs> DelCovAreaClicked;

}

}

Классы Presentor’ов

namespace TicketReservationSystem

{

public class TransactionsPresenter

{

IForm2 view;

public TransactionsPresenter(IForm2 vw)

{

view = vw;

view.AddTransactionClicked += new EventHandler<EventArgs>(Add);

}

public void Add(object sender, EventArgs e)

{

//Обращаемся к модели

TransactionsModel tranMod = new TransactionsModel();

tranMod.NewTransaction(view.Id\_agreement, view.Operation\_type,

view.Transaction\_amount, view.Transaction\_date);

}

}

}

namespace TicketReservationSystem

{

public class CovAreasPresenter

{

IForm6 view;

public CovAreasPresenter(IForm6 vw)

{

view = vw;

view.AddCovAreaClicked += new EventHandler<EventArgs>(Add);

view.DelCovAreaClicked += new EventHandler<EventArgs>(Del);

}

public void Add(object sender, EventArgs e)

{

//Обращаемся к модели

CovAreasModel covAreaMod = new CovAreasModel();

covAreaMod.NewCovArea(view.District, view.Street, view.Num\_house);

}

public void Del(object sender, EventArgs e)

{

//Обращаемся к модели

CovAreasModel covAreaMod = new CovAreasModel();

covAreaMod.DelCovArea(view.idCovArea);

}

}

}

Классы модели (представлен один)

namespace TicketReservationSystem

{

public class TransactionsModel

{

public void NewTransaction(int Id\_agreement, string Operation\_type,

int Transaction\_amount, DateTime Transaction\_date)

{

PROVIDER\_DBDataContext dc = new PROVIDER\_DBDataContext();

var Trans = new transactions()

{

id\_agreement = Id\_agreement,

operation\_type = Operation\_type,

transaction\_amount = Transaction\_amount,

transaction\_date = Transaction\_date

};

dc.transactions.InsertOnSubmit(Trans); // добавление в БД

dc.SubmitChanges(); // выполняет соответствующие команды

// для внесения изменений базы данных

}

}

}

Класс формы (представлен один)

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TicketReservationSystem

{

public partial class Form2 : Form, IForm2

{

PROVIDER\_DBDataContext dc;

int idAgreement;

public event EventHandler<EventArgs> AddTransactionClicked; //событие для наблюдателя

public Form2(int idAgr)

{

InitializeComponent();

dc = new PROVIDER\_DBDataContext();

idAgreement = idAgr;

fillTransactions();

transaction\_date.Enabled = false;

}

//Получаем данные транзакции

public int Id\_agreement { get { return Convert.ToInt32(id\_agreement.Text); } }

public string Operation\_type { get { return operation\_type.Text; } }

public int Transaction\_amount { get { return (int) transaction\_amount.Value; } }

public DateTime Transaction\_date { get { return transaction\_date.Value; } }

//ОТОБРАЖЕНИЕ ДАННЫХ ВЫБРАННОЙ ТРАНЗАКЦИИ

public void transactions\_list\_SelectionChanged(object sender, EventArgs e)

{

//Если строка не пустая

try

{

if (transactions\_list.CurrentRow.Index >= 0)

{

//При выделении строчки в таблице

int idt = Convert.ToInt32(transactions\_list.Rows[transactions\_list.CurrentRow.Index].Cells[0].Value.ToString());

var t = dc.transactions.SingleOrDefault(c => c.transaction\_id == idt);

if (t != null)

{

operation\_type.Text = t.operation\_type;

transaction\_amount.Value = t.transaction\_amount;

transaction\_date.Value = t.transaction\_date;

}

}

}

catch { }

}

// ЗАПОЛНЕНИЕ ТАБЛИЦЫ

private void fillTransactions()

{

// Отметим код договора

id\_agreement.Text = idAgreement.ToString();

// Перезаполним таблицу

int countRow = transactions\_list.Rows.Count;

for (int i = 0; i < countRow; i++)

{

transactions\_list.Rows.RemoveAt(0);

}

var query2 = dc.transactions;

int numRow = 0;

foreach (var item in query2)

{

transactions\_list.Rows.Add();

transactions\_list.Rows[numRow].Cells[0].Value = item.transaction\_id.ToString();

transactions\_list.Rows[numRow].Cells[1].Value = item.id\_agreement.ToString();

transactions\_list.Rows[numRow].Cells[2].Value = item.operation\_type.ToString();

transactions\_list.Rows[numRow].Cells[3].Value = item.transaction\_amount.ToString();

transactions\_list.Rows[numRow].Cells[4].Value = item.transaction\_date.ToString();

numRow++;

}

}

// ДОБАВЛЕНИЕ НОВОЙ ТРАНЗАКЦИИ

public void btn\_transaction\_add\_Click(object sender, EventArgs e)

{

if (operation\_type.Text != "" && transaction\_amount.Value != 0 /\*&& transaction\_date.Value >= System.DateTime.Now\*/)

{

// НАБЛЮДАТЕЛЬ

AddTransactionClicked(this, e);

// обновим таблицу

fillTransactions();

}

else

{

MessageBox.Show("Не все поля заполнены верно!");

}

}

}

}