Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

ОТЧЕТ

К лабораторной работе №2 (Разработка бизнес-логики приложения)

по дисциплине «Системное программное обеспечение»

Выполнил студент

группы 538:

\_\_\_\_\_\_\_\_ Черников А.И.

Проверил

доцент кафедры КСУП:

\_\_\_\_\_\_\_\_ Коцубинский В.П.

Дата: «\_\_\_»\_\_\_\_\_\_\_\_2020 г.

Томск 2020

Оглавление

[**1 Цель и задачи лабораторной работы** 3](#_Toc53765365)

[**2 Ход лабораторной работы** 4](#_Toc53765366)

[**3 Заключение** 6](#_Toc53765367)

**1 Цель и задачи лабораторной работы**

Цель работы: изучить типовые требования, предъявляемые к бизнес-логике приложения, получить умения разработки логики приложения с обеспечением данных требований.

Задачи:

1. Изучить требования и процесс разработки логики приложения.

2. Повторить синтаксис языка C# для разработки объектно-ориентированных программ.

3. Разработать классы, необходимые для работы логики приложения.

4. Обеспечить целостность данных классов с помощью свойств и механизма генерации исключений.

**2 Ход лабораторной работы**

В ходе лабораторной работы мы должны реализовать следующие типы данных:

-Перечисление «Категория заметки», содержащее значения «Работа», «Дом», «Здоровье и Спорт», «Люди», «Документы», «Финансы», «Разное».

-Класс «Заметка» с полями «Название», «Категория заметки», «Текст заметки», «Время создания», «Время последнего изменения». Название, категория и текст заметки доступны для изменений. Время создания инициализируется один раз при создании объекта «Заметка» и больше не модифицируется, доступна для чтения. Время последнего изменения меняется автоматически при изменении названия, категории или текста заметки. Название ограничено 50 символами. Название по умолчанию «Без названия». Допустимы заметки с одинаковыми названиями. Реализует интерфейс ICloneable.

-Класс «Проект». Содержит список (или словарь) всех заметок, созданных в приложении.

-Класс «Менеджер проекта». Реализует метод для сохранения объекта «Проект» в файл и метод загрузки проекта из файла. Сохранение и загрузка осуществляются в один и тот же файл«...\MyDocuments\NoteApp.notes», имя которого задано закрытой константой внутри класса.Формат данных – json, библиотека сериализации (преобразования данных в json-формат) –NewtonsoftJSON.NET

UML-диаграмма классов, составленная в Enterprise Architect, представлена на рисунке 2.1:

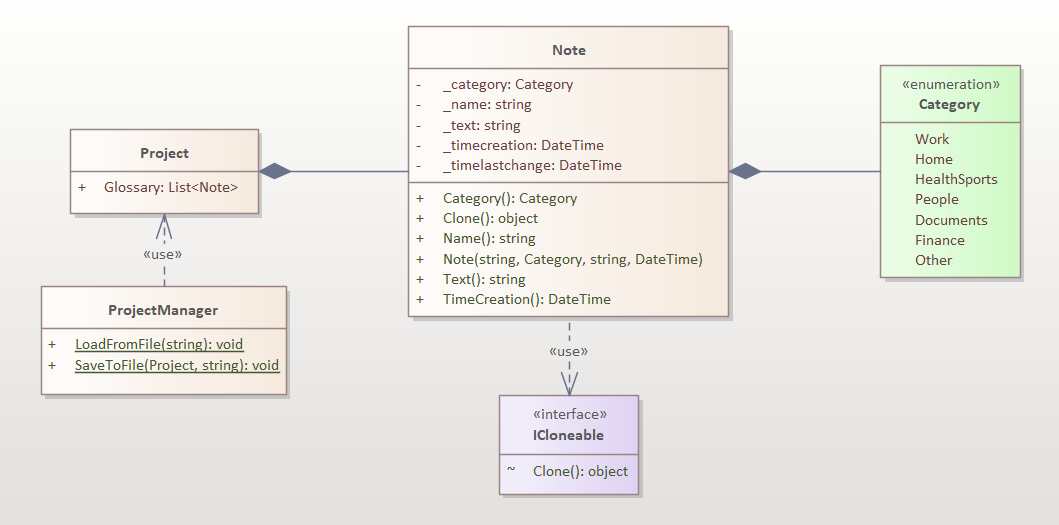


Рисунок 2.1 - UML-диаграмма классов.

Дальше создаем сами классы в Visual Studio:

Код класса Note.cs:

using System;

namespace NoteApp

{

/// <summary>

/// Класс заметки, хранящий информацию о заголовке, категории, тексте и времени создания и изменении

/// </summary>

public class Note : ICloneable

{

private string \_name;

private Category \_category;

private string \_text;

private DateTime \_timecreation;

private DateTime \_timelastchange;

/// <summary>

/// Возвращает и задает название заметки

/// </summary>

public string Name

{

get {return \_name;}

set

{

\_name = value;

if (\_name.Length > 50)

{

throw new Exception("Длина названия не должна превышать 50 символов");

}

if (\_name.Length == 0)

{

\_name = "Без названия";

}

}

}

/// <summary>

/// Возвращает и задает категорию заметки

/// </summary>

public Category Category

{

get { return \_category;}

set

{

\_category = value;

}

}

/// <summary>

/// Возвращает и задает текст заметки

/// </summary>

public string Text

{

get { return \_text; }

set

{

\_text = value;

}

}

/// <summary>

/// Возвращает время создания заметки

/// </summary>

public DateTime TimeCreation { get; private set; }

/// <summary>

/// Возвращает время изменения заметки

/// </summary>

public DateTime TimeLastChange

{

get { return \_timelastchange; }

set

{

\_timelastchange = value;

}

}

/// <summary>

/// Создание заметки

/// </summary>

public Note(string name,Category category,string text, DateTime timecreation)

{

Name = name;

Category = category;

Text = text;

TimeCreation = timecreation;

}

/// <summary>

/// Клонирование заметки

/// </summary>

public object Clone()

{

return new Note(this.Name, this.Category, this.Text, this.TimeCreation)

{

TimeLastChange = this.TimeLastChange

};

}

}

}

Код перечисления Category.cs:

public enum Category

{

Work,

Home,

HealthSports,

People,

Documents,

Finance,

Other

}

Код класса Project.cs:

public class Project

{

public List<Note> Glossary=new List<Note>();

}

Код класса ProjectManager.cs:

public class ProjectManager

{

public static void SaveToFile(Project data, string file)

{

JsonSerializer serializer = new JsonSerializer

{

Formatting = Formatting.Indented,

TypeNameHandling = TypeNameHandling.All

};

using (StreamWriter sw = new StreamWriter(file))

using (JsonWriter writer = new JsonTextWriter(sw))

{

serializer.Serialize(writer, data);

}

}

public static Project LoadFromFile(string file)

{

JsonSerializer serializer = new JsonSerializer

{

Formatting = Formatting.Indented,

TypeNameHandling = TypeNameHandling.All

};

using (StreamReader sr = new StreamReader(file))

using (JsonReader reader = new JsonTextReader(sr))

{

return (Project)serializer.Deserialize<Project>(reader);

}

}

}

Код в Form1.cs:

public partial class Form1 : Form

{

public static string \_filePath;

public static string \_fileName = "json.txt";

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

\_filePath = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);

Project p=new Project();

Note[] note = new Note[5];

for (int i = 0; i < 5; i++)

{

note[i]=new Note("1"+i,Category.Work,"aaaa",DateTime.Today);

p.Glossary.Add(note[i]);

}

ProjectManager.SaveToFile(p, \_filePath + @"\" + \_fileName);

p = null;

p = ProjectManager.LoadFromFile(\_filePath + @"\" + \_fileName);

}

}

Проверим, что в файл записан наш лист с заметками:

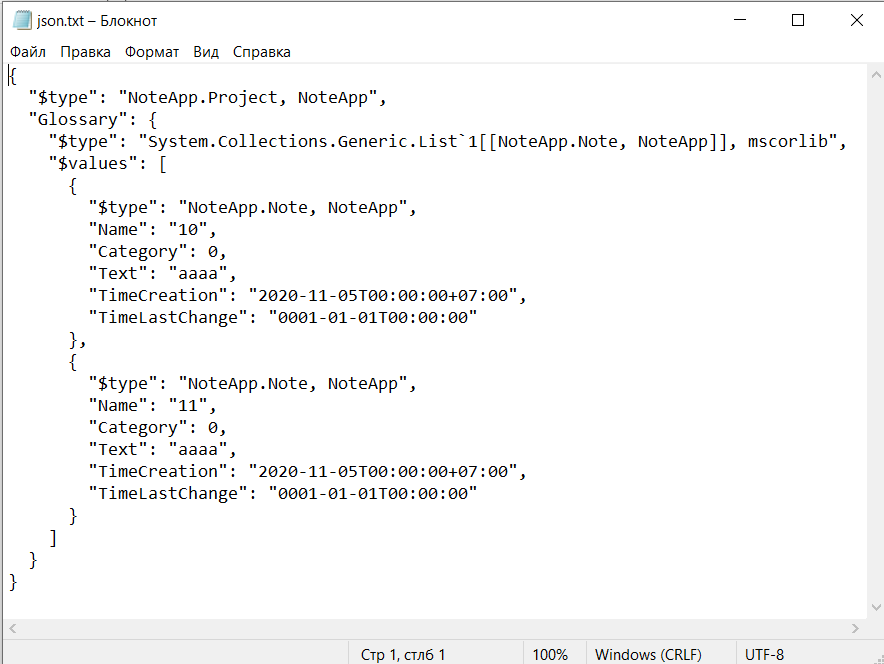


Рисунок 2.2 – Файл с заметками.

После просмотрим историю коммитов ветки develop, чтобы убедиться, что мы все сделали верно:

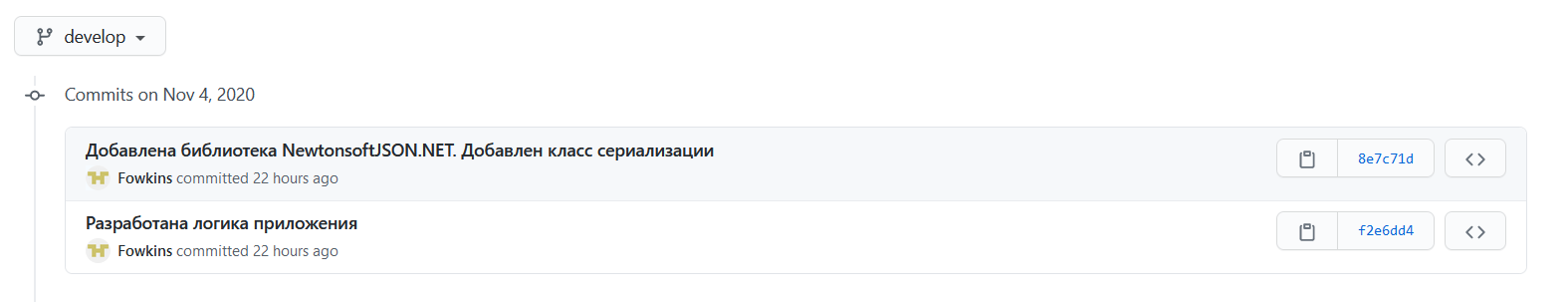


Рисунок 2.3 - История коммитов ветки develop.

# **3 Заключение**

В ходе данной лабораторной работы мы изучили процесс разработки логики приложений, повторили синтаксис языка C#, разработали классы, необходимые для работы логики приложения и воспользовались механизмом генерации исключений.

Задачи на лабораторную работы выполнены.