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:

\_\_\_\_\_\_\_\_ Асмоловский А.В.

Проверил

доцент кафедры КСУП:

\_\_\_\_\_\_\_\_ Коцубинский В.П.

Дата: «\_\_\_»\_\_\_\_\_\_\_\_2020 г.

Томск 2020

Оглавление

[**1 Цель и задачи лабораторной работы** 3](#_Toc56606449)

[**2 Ход лабораторной работы** 4](#_Toc56606450)

[**3 Заключение** 11](#_Toc56606451)

**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 проектаиз 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 Dio, представлена на рисунке 2.1:

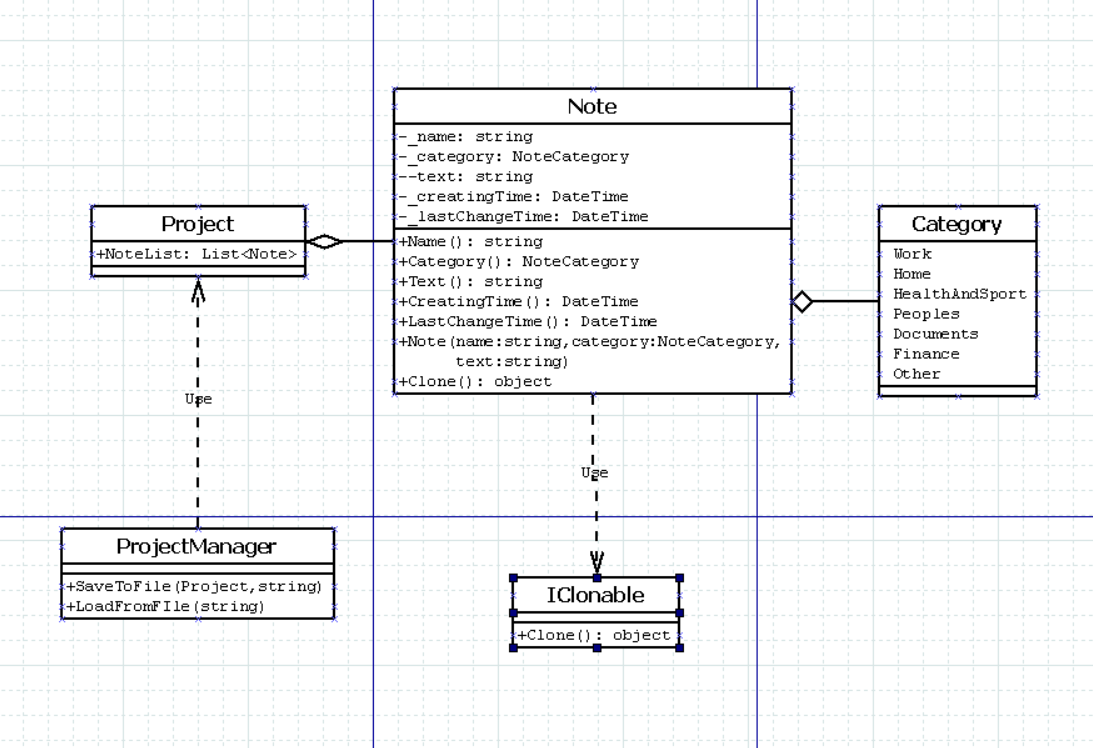


Рисунок 2.1 — UML диаграмма классов.

Дальше создаем сами классы в VisualStudio:

Код класса Note.cs:

using System;

using System.Collections.Generic;

using System.Dynamic;

using System.Text;

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 NoteCategory \_category;

private string \_text;

private DateTime \_creatingTime;

private DateTime \_lastChangeTime;

/// <summary>

/// Возвращает и задает название заметки

/// </summary>

public string Name

{

get { return \_name; }

set

{

\_name = value;

if (value.Length == 0)

{

\_name = "NoName";

return;

}

if (value.Length>50)

{

throw new ArgumentException("Длинна названия не может превышать 50 символов");

}

}

}

/// <summary>

/// Возвращает и задает категорию заметки

/// </summary>

public NoteCategory 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 и задает время создания заметки

/// </summary>

public DateTime CreatingTime

{

get { return \_creatingTime; }

private set { \_creatingTime = value; }

}

/// <summary>

/// Возвращает и задает время последнего изменения заметки

/// </summary>

public DateTime LastChangeTime

{

get { return \_lastChangeTime; }

set { \_lastChangeTime = value; }

}

/// <summary>

/// Создание заметки

/// </summary>

public Note(string name, NoteCategory category, string text)

{

Name = name;

Category = category;

Text = text;

CreatingTime = DateTime.Now;

}

public Note()

{

}

public object Clone()

{

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

{

LastChangeTime = this.LastChangeTime

};

}

}

}

Код перечисления Category.cs:

/// <summary>

/// перечисление категорий заметок

/// </summary>

public enum NoteCategory

{

Work,

Home,

HealthAndSport,

Peoples,

Documents,

Finance,

Other

}

Код класса Project.cs:

namespace NoteApp

{

/// <summary>

/// Класс заметок, содержащий список всех заметок.

/// </summary>

public class Project

{

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

}

}

Код класса ProjectManager.cs:

namespace NoteApp

{

public class ProjectManager

{

/// <summary>

/// константа содержащая путь

/// </summary>

private static string \_path = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile) + @"\source\repos\Dronee44\NoteApp\NoteApp\Test\bin\Debug\netcoreapp3.1\NoteApp.json";

public static void SaveToFile(Project data, string file)

{

JsonSerializer serializer = new JsonSerializer

{

Formatting = Formatting.Indented,

TypeNameHandling = TypeNameHandling.All

};

////Открываем поток для записи в файл с указанием пути

using (StreamWriter sw = new StreamWriter(file))

using (JsonWriter writer = new JsonTextWriter(sw))

{

//Вызываем сериализацию и передаем объект, который хотим сериализовать

serializer.Serialize(writer, data);

}

}

/// <summary>

/// Сохранения списка заметок в путь по умолчанию

/// </summary>

/// <param name="data">список заметок </param>

public static void SaveToFile(Project data)

{

SaveToFile(data, \_path);

}

public static Project LoadFromFile(string file)

{

JsonSerializer serializer = new JsonSerializer

{

Formatting = Formatting.Indented,

TypeNameHandling = TypeNameHandling.All

};

//Открываем поток для чтения из файла с указанием пути

using (StreamReader sr = new StreamReader(file))

using (JsonReader reader = new JsonTextReader(sr))

{

//Вызываем десериализацию и явно преобразуем результат в целевой тип данных

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

}

}

/// <summary>

/// Загрузка списка из файла в путь по умолчанию

/// </summary>

/// <returns>Возвращает список заметок</returns>

public static Project LoadFromFile()

{

return LoadFromFile(\_path);

}

}

}

Код класса Test.cs:

namespace NoteApp

{

class Test

{

static void Main(string[] args)

{

Note note = new Note("Zametka 1",NoteCategory.Work,"Vse rabotaet");

Note note2 = new Note("Zametka 2", NoteCategory.Peoples, "Opat vse rabotaet");

Console.WriteLine(note.Name + " " + note.Text + " " + note.Category);

Console.WriteLine(note2.Name + " " + note2.Text + " " + note2.Category);

Project prj= new Project();

Project prj2 = new Project();

prj.NoteList.Add(note);

prj.NoteList.Add(note2);

ProjectManager.SaveToFile(prj);

//prj = null;

//prj = ProjectManager.LoadFromFile("NoteApp.txt");

Note note3 = new Note("keka", NoteCategory.Finance, "Normas Zakladka");

Console.WriteLine(note3.CreatingTime);

}

}

}

Проверим, что в файл записан наш лист с заметками:

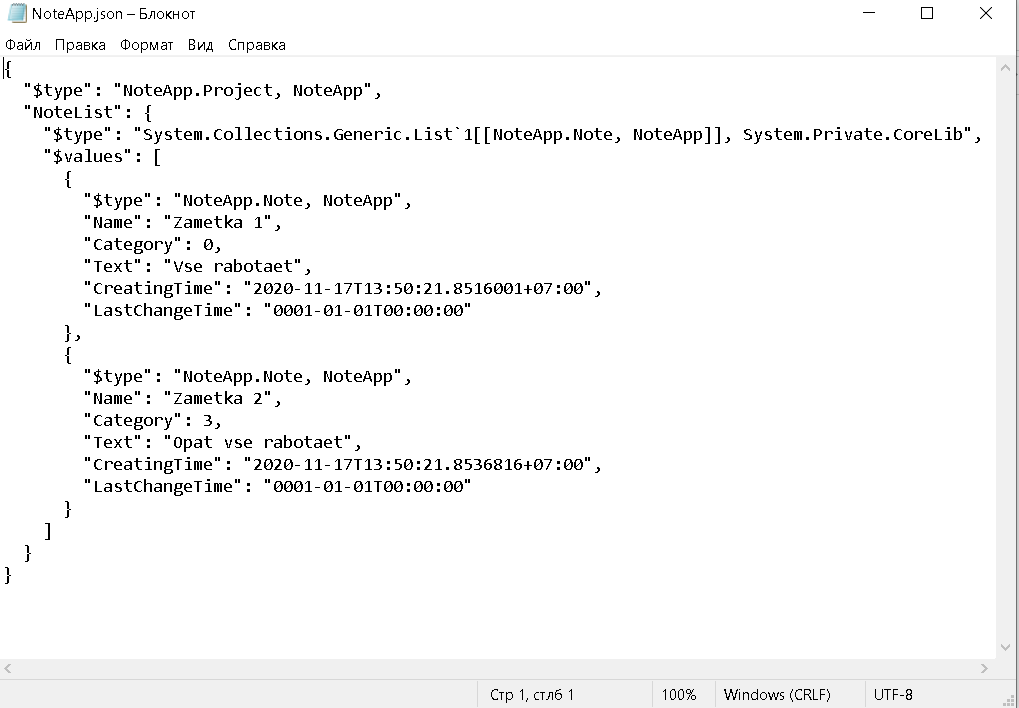


Рисунок 2.2 — Файл с заметками.

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

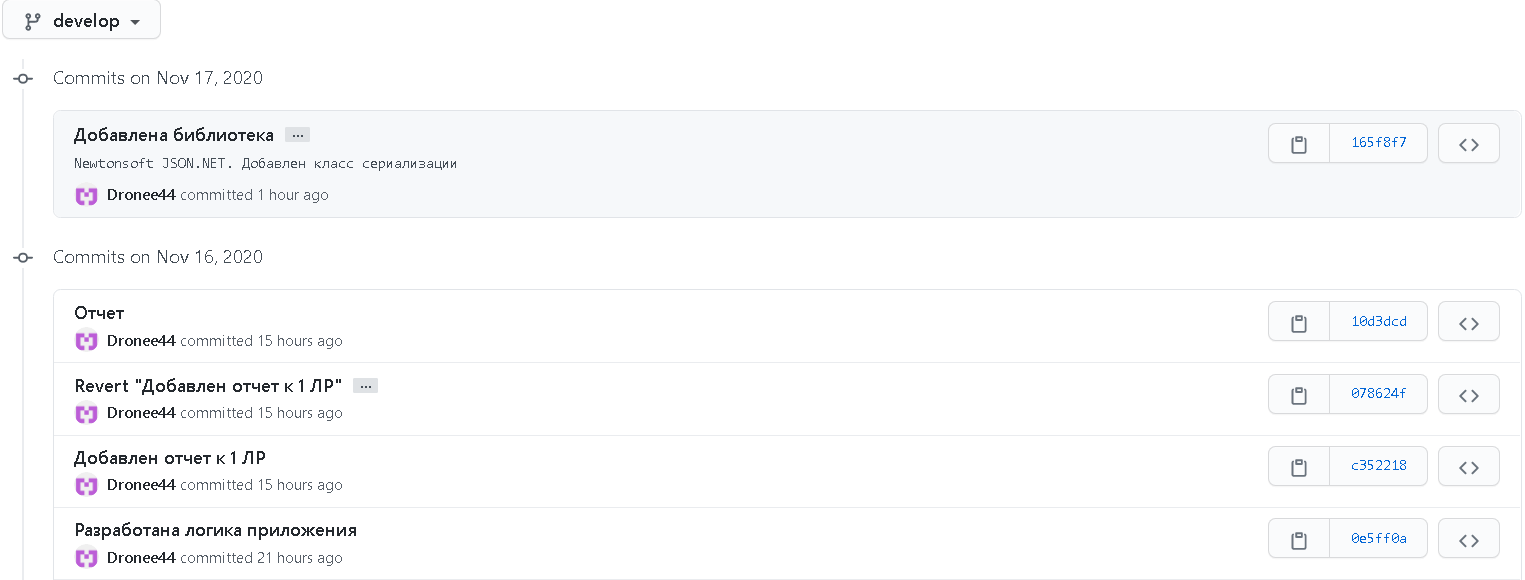


Рисунок 2.3 — История коммитов ветки develop.

# **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.