Министерство образования и науки 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»

Студент

гр. 538 А. В. Вачиля

Принял:

доцент кафедры КСУП:

\_\_\_\_\_\_\_\_ Коцубинский В.П.

Дата: «\_\_\_»\_\_\_\_\_\_\_\_2020 г.

2020

Оглавление

[1 Цель и задачи лабораторной работы 3](#_Toc57161285)

[2 Ход лабораторной работы 4](#_Toc57161286)

[3 Заключение 12](#_Toc57161287)

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**

Начинаем лабораторную работы с реализации следующих типов данных:

* Перечисление «Категория», содержащее значения категорий заметок «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«...\Desktop\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:

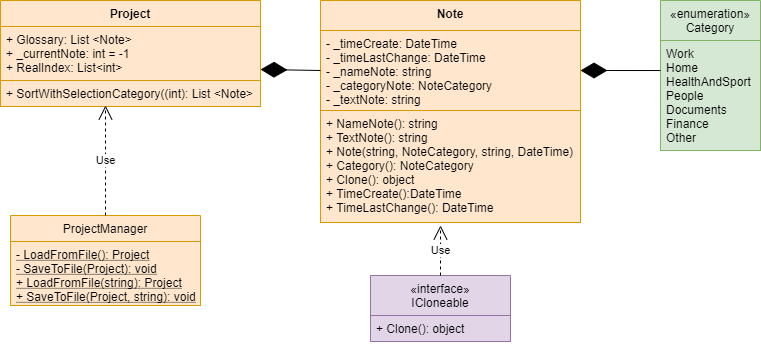


Рисунок 2.1 – UML-диаграмма классов

Далее создаём сами классы

Код перечисления Category.cs:

/// <summary>

/// Перечисление, содержащее категории записей: Работа, Дом, Здоровье и спорт, Люди, Документы, Финансы, Разное

/// </summary>

public enum Category

{

Work,

Home,

HealthSports,

People,

Documents,

Finance,

Other

}

Код класса Note.cs:

using System;

namespace NoteApp

{

/// <summary>

/// Класс, содержащий поля заметок

/// </summary>

public class Note : ICloneable

{

private string \_nameNote;

private Category \_categoryNote;

private string \_texNotet;

private DateTime \_timeCreate;

private DateTime \_timeLastChange;

/// <summary>

/// Метод ввода и получения названия заметки

/// </summary>

public string Name

{

get {return \_name;}

set

{

\_nameNote = value;

if (\_nameNote.Length > 50)

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

if (\_nameNote.Length == 0)

\_nameNote = "Без названия";

} }

/// <summary>

/// Метод ввода и получения категорию заметки

/// </summary>

public NoteCategory Category

{

get { return \_categoryNote; }

set { \_categoryNote = value; }

}

/// <summary>

/// Метод ввода и получения текста заметки

/// </summary>

public string Text

{

get { return \_text; }

set { \_text = value; }

}

/// <summary>

/// Метод фиксирования и получения времени создания записи

/// </summary>

public DateTime TimeCreate

{

get { return \_timeCreate; }

private set { \_timeCreate = value; }

}

/// <summary>

/// Метод фиксирования и получения времени изменения записи

/// </summary>

public DateTime TimeLastChange

{

get { return \_timeLastChange; }

set { \_timeLastChange = value; }

}

/// <summary>

/// Конструктор заметки

/// </summary>

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

{

NameNote = name;

Category = category;

TextNote = text;

TimeCreate = timecreate;

}

/// <summary>

/// Реализация интерфейса клонирования IClonable

/// </summary>

public object Clone()

{

return new Note(this.NameNote, this.Category, this.TextNote, this.TimeCreate)

{

TimeLastChange = this.TimeLastChange

};

}

}

}

Код класса Project.cs:

/// <summary>

/// Содержит список(или словарь) всех Созданных заметок

/// </summary>

public class Project

{

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

/// <summary>

/// Индекс текущей заметки

/// </summary>

public int \_currentNote = -1;

//Сортировка категорий

public List<int> RealIndexes = new List<int>();

public List<Note> SortWithSelectionCategory(int category)

{

var sortNotes = new List<Note>();

//если выбрана категория All

if (category == 0)

{

RealIndexes.Clear();

for (int i = 0; i < Glossary.Count; i++)

{

sortNotes.Add(Glossary[i]);

RealIndexes.Add(i);

}

}

//если другая категория

else

{

RealIndexes.Clear();

for (int i = 0; i < Glossary.Count; i++)

{

if ((int)Glossary[i].Category == category - 1)

{

sortNotes.Add(Glossary[i]);

RealIndexes.Add(i);

}

}

}

return sortNotes;

}

}

Код класса ProjectManager.cs:

public class ProjectManager

{

private static string \_path = Environment.GetFolderPath(Environment.SpecialFolder.Desktop) + @"\NoteApp.notes"; //Закрытая константа, содержащая путь

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);

}

}

/// <summary>

/// Сохранение 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

};

using (StreamReader sr = new StreamReader(file))

using (JsonReader reader = new JsonTextReader(sr))

{

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);

}

}

Можем убедиться, что программа правильно работает можно запустив её и открыв созданный файл с заметками:

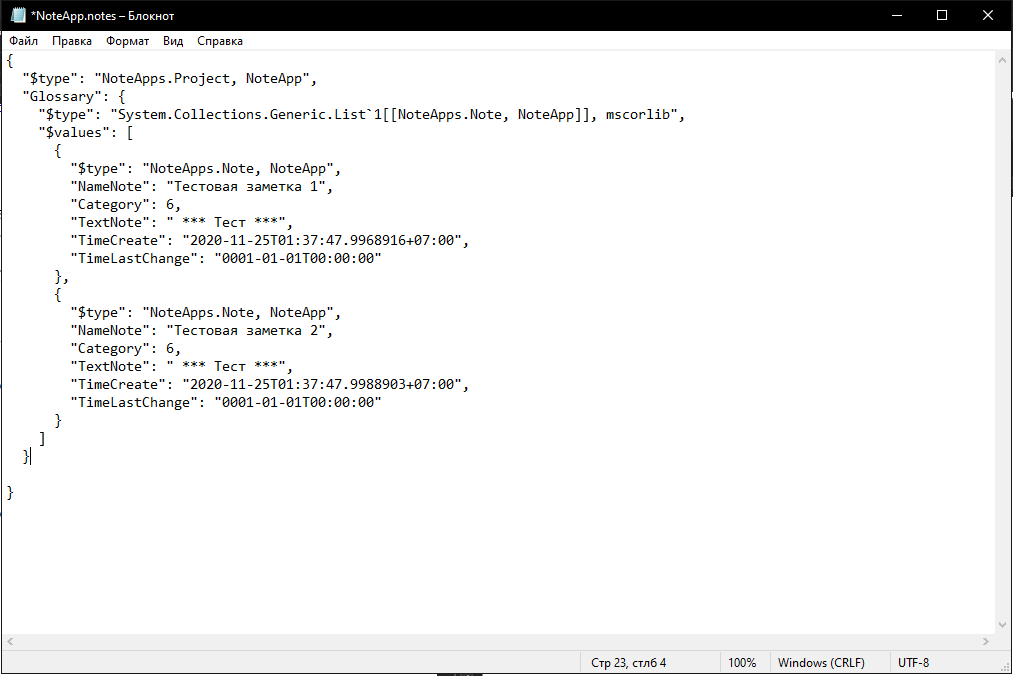


Рисунок 2.2 – Файл с заметками

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

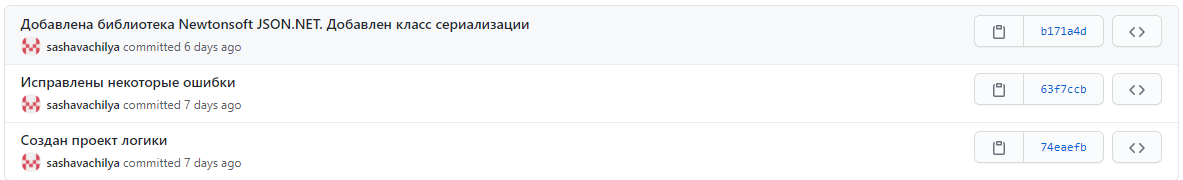


Рисунок 2.3 – История коммитов ветки develop

3 Заключение

В ходе данной лабораторной работе мы изучили процесс проектирования и разработки логики приложения, разработали классы, необходимые для работы логики приложения и воспользовались механизмом генерации исключения. Цель и все поставленные задачи были успешно выполнены.