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)

РАЗРАБОТКА БИЗНЕС-ЛОГИКИ ПРИЛОЖЕНИЯ

Отчет по лабораторной работе №2 по дисциплине «Новые технологии в программировании»

Студент гр. 588-1

_____/ Севостьянов К.С.
(подпись)
«___»_____20__г.

Руководитель старший научный сотрудник, доцент каф. КСУП

_____/ Горяинов А.Е.
(подпись)
«___»_____20__г.

Томск 2021

Оглавление

Введение

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 генерации исключений.

2 Особенности разработки логики приложения

Исполняемое объектно-ориентированное приложение представляет множество ***объектов***, взаимодействующих между собой с помощью ***сообщений***. Каждый объект является экземпляром некоторого ***класса***. Одному классу может принадлежать множество объектов. Класс определяет атрибуты класса – его ***поля*** – и спецификацию сообщений, которые ему можно послать – ***методы*** класса.

Интерфейс класса – это набор методов класса, доступных для использования другими классами. Поля класса предпочтительно помещать под модификатор доступа **private** для защиты целостности данных, а доступ к их значениям предоставлять с помощью специальных методов: геттеров и сеттеров. Методы класса можно разделить на следующие группы:

1) конструкторы;

2) деструкторы;

3) геттеры;

4) сеттеры;

5) операционные.

Конструкторы – методы класса, вызываемые для создания экземпляра класса и содержание инструкции по инициализации объекта.

Деструкторы – методы, вызываемые при уничтожении объектов класса; как правило, содержат инструкции для освобождения динамической памяти, используемой внутри объекта.

Геттеры – методы, позволяющие получить текущие значения полей класса.

Сеттеры – методы, позволяющие установить новые значения в поля класса.

Операционные – методы, выполняющие обработку текущего состояния объекта или входных данных метода.

Виртуальные функции – функции класса, реализация которых может быть переопределена в дочерних классах. Чисто виртуальные функции – это виртуальные функции, не имеющие реализации в базовом классе. Класс, который имеет хотя бы одну чисто виртуальную функцию, называется абстрактным.

Полиморфизм (в ООП) – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Для полиморфизма необходимо создать базовый класс с чисто виртуальными функциями – интерфейс. В дочерних классах определяется собственная реализация интерфейса. Таким образом, можно сказать, что дочерние классы обладают единым интерфейсом. В дальнейшем, вне иерархии наследования создается указатель на базовый класс, который может хранить экземпляры дочерних полиморфных классов.

Механизм обработки исключений необходим для раннего обнаружения ошибок в исходном коде и обеспечения целостности данных в классе. Для того, чтобы защитить классы от некорректных значений, используется механизм выбрасывания исключений для прерывания программы.

Выбрасывание исключения выполняется с помощью ключевого слова throw. Оператор throw аналогичен оператору return – инструкции метода после этого оператора не выполняются, а управление возвращается в точку вызова. Однако, в отличие от return, если в точке вызова нет обработки исключения (операторов try-catch), то программа аварийно завершится. Оператор throw может выбросить объект любого типа данных, в том числе int, double, string и т.д. Однако следует выбрасывать объекты класса Exception и его наследников.

Требования к бизнес-логике.

Необходимо реализовать следующие типы данных:

- Класс «Номер телефона» с полем «Номер». Поле должно быть числовым и содержать ровно 11 цифр. Первая цифра должна быть ‘7’ (только Российские телефонные номера)

- Класс «Контакт» с полями «Фамилия», «Имя», «Номер телефона», «Дата рождения», «e-mail», «ID ВКонтакте». Все поля доступны для изменений. Фамилия, имя и e-mail ограничены 50 символами каждое, ID Вконтакте ограничен 15 символами. Первая буква в фамилии и имени должна преобразовываться к верхнему регистру. Дата рождения не может быть более текущей даты и не может быть менее 1900 года. Допустимы контакты с одинаковыми фамилиями и именами. Реализует интерфейс ICloneable.

- Класс «Проект». Содержит список(или словарь) всех контактов, 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 «…\My Documents\ContactsApp.notes», имя которого задано закрытой константой внутри класса. Формат данных – json, библиотека сериализации (преобразования данных в json-формат) – Newtonsoft JSON.NET.

3 Классы проекта

1. **Класс «Номер телефона».**

Класс «Номер телефона» с полем «Номер». Поле должно быть числовым и содержать ровно 11 цифр. Первая цифра должна быть ‘7’ (только Российские телефонные номера)

2. **Класс «Контакт».**

Класс «Контакт» с полями «Фамилия», «Имя», «Номер телефона», «Дата рождения», «e-mail», «ID ВКонтакте». Все поля доступны для изменений. Фамилия, имя и e-mail ограничены 50 символами каждое, ID Вконтакте ограничен 15 символами. Первая буква в фамилии и имени должна преобразовываться к верхнему регистру. Дата рождения не может быть более текущей даты и не может быть менее 1900 года. Допустимы контакты с одинаковыми фамилиями и именами. Реализует интерфейс ICloneable.

3. **Класс «Проект» («Project»).**

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

4. **Класс «Менеджер проекта» («ProjectManager»).**

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 «…\My Documents\ContactsApp.notes», имя которого задано закрытой константой внутри класса. Формат данных – json, библиотека сериализации (преобразования данных в json-формат) – Newtonsoft JSON.NET.

UML-диаграмма классов представлена на рисунке 3.1.

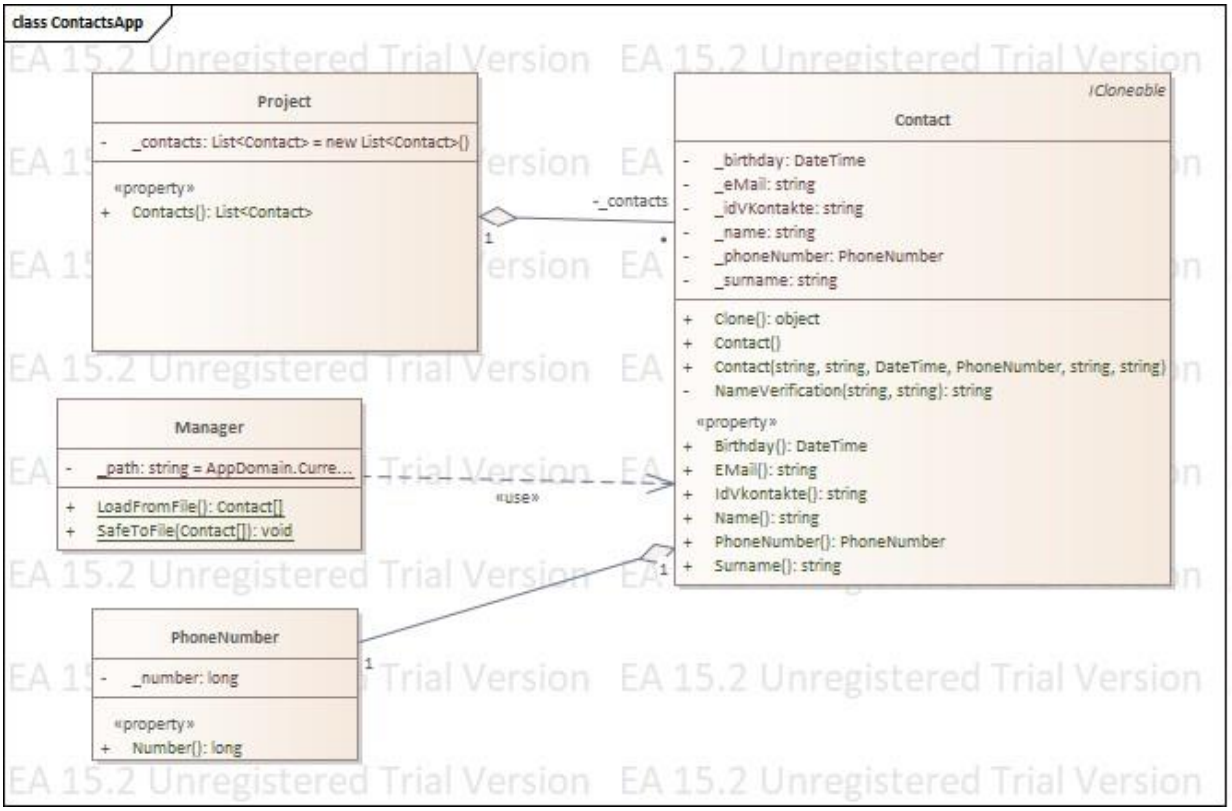

Рисунок 3.1 – UML-диаграмма классов проекта логики

4 Заключение

В ходе лабораторной работы изучены типовые требования, предъявляемые к бизнес-логике приложения, получены умения разработки логики приложения с обеспечением данных требований.