

## **Лабораторная работа 3**

### **Создание простейшей информационной системы (часть 1)**

**Цель работы:** разработать информационную систему (ИС), позволяющую осуществлять ввод, хранение и простейшую обработку данных предметной области. Хранение накопленных данных осуществляется посредством xml.

В ходе работы будет создано приложения Windows Forms, позволяющее открывать созданный XML файл, считывать информацию из файла в коллекцию и выводить информацию на форму в компонент DataGridView.

Кроме этого, добавим в приложение вторую форму, позволяющую вводить составные данные (например, адрес). И, в продолжении темы разработки качественного пользовательского интерфейса, добавим проверку на правильность ввода некоторых значений в DataGridView: чисел и даты.

### **Часть 1. Введение в элементы XML**

#### **Теоретические сведения**

Описание XML- документа представляет собой простой текст, который можно набрать в любом текстовом редакторе. Создайте в текстовом редакторе Notepad новый файл и введите текст XML-документа, сохранив с расширением .xml

Каждый XML- документ размечается тегам. **Тег** - это текст, заключенный в угловые скобки, который не относится к содержанию документа, а отмечает начало или конец какого-либо документа.

**Элемент** документа XML имеет начальный тег, задающий имя элемента и дополнительную информацию (атрибуты), и конечный тег, содержащий то же имя элемента со знаком "/" впереди. Элементы определяют логическую структуру документа и несут в себе информацию, содержащуюся в документе. Имя элемента считается так же его типом.

Элементы XML-документа могут быть вложенными.

Пока не задан формат отображения XML-документа на экране, браузер применяет способ, принятый по умолчанию (IE будет использовать встроенную таблицу стилей для отражения документа). Одним из вариантов указания способа отображения документа является создание для него таблицы каскадных стилей (CSS).

**Правильно оформленными (well-formed) XML-документом** называется документ, удовлетворяющий минимальному набору правил соответствия для XML-документа.

Правильно оформленный XML-документ состоит из двух основных частей: **пролога** и **корневого элемента**. Помимо этого, он может содержать **комментарии, инструкции и пробелы**. Корневой элемент может содержать вложенные элементы. Элементы должны быть правильно вложены. Если элемент начинается внутри некоторого другого элемента, то и заканчиваться он должен внутри того же элемента.

Каждый элемент состоит из начального тега, содержимого и конечного тега. Исключением является пустой элемент, для которого может использоваться единственный тег пустого элемента `<Emptyelement/>`.

Правила использования имен элементов (**типов элемента**):

- имя должно начинаться с буквы или символа подчеркивания (`_`),
- следующие после первого символы могут быть буквами, цифрами, точкой, тире или подчеркиванием.
- Не следует использовать имена, начинающиеся с префикса "xml"(в любом сочетании строчных или прописных букв).
- Имя, записанное в начальном теге, должно в точности соответствовать имени в конечном теге;
- Соблюдение регистра существенно для имен элементов, как и для всего текста в описании разметки.

**Содержимым элемента** считается текст, расположенный между начальным и конечным тегами.

В начальный тег элемента либо в тег пустого элемента можно включить один или несколько описаний атрибутов. Описание атрибута представляет собой пару **имя = значение**, связанную с данным элементом. Каждое имя атрибута может только один раз присутствовать в начальном теге элемента. Правила именования атрибутов аналогичны правилам именования элементов. Задание атрибутов обеспечивает альтернативный способ включения информации в элемент. Значение, которое можно присваивать атрибуту, представляет собой строку символов (литерал), ограниченных одинарными или двойными кавычками.

**Комментарий** начинается с символов `<!--` и заканчивается символами `-->`. Внутри комментариев не может содержаться двойное тире (`--`), символ левой угловой скобки (`<`) и знак амперсанда (`&`). Комментарии можно вставлять в любое место XML-документа, кроме описания тега.

**Инструкции** (инструкции по обработке) предназначены для XML-процессора, который будет обрабатывать документ. Общий вид инструкции:

### <?наименование данные?>

**Наименование** указывает получателя инструкции. Параметр **данные** задает содержание инструкции. По наименованию конкретный XML-процессор определяет, предназначены данные ему или другому процессору. Инструкции могут помещаться в любое место XML-документа вне описания тегов.

Инструкция `<?xml-stylesheet type="text/css" href="file_2.css"?>` предписывает Internet Explorer 5 использовать CSS-таблицу из файла `file_2.css`.

Внутри символьных данных в содержимом элемента нельзя помещать некоторые специальные символы (<, &, :), так как это может привести к путанице при обработке документа. Одним из возможных путей преодоления этих ограничений является использование **разделов символьных данных**. Такой раздел начинается с символов `<![CDATA[` и заканчивается `]]>`. Все символы внутри раздела CDATA рассматриваются как литеральная часть символьных данных элемента, а не как XML-разметка. Раздел CDATA может располагаться в любом месте документа, занимаемом символьными данными. Разделы CDATA не могут быть вложенными.

Документ XML может содержать **пустые строки**, состоящие из одного или нескольких пробелов, символов табуляции, символа Enter. Можно свободно добавлять пробелы и переводы строк между:

- начальными и конечными тегами;
- комментариями;
- инструкциями по обработке.

Недолжно быть пробелов между открывающейся угловой скобкой и именем элемента.

### Практическая часть. Создание документа XML

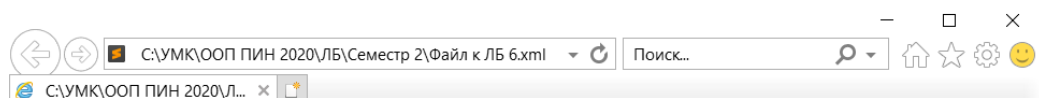
Проработайте ниже предложенный пример создания XML-документа и его отображения с помощью каскадных таблиц стилей.

Создайте в текстовом редакторе Notepad (или Sublime Text 3) новый файл и введите текст XML-документа, сохранив с расширением `.xml`.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Имя файла Файл к ЛБ 6.xml -->
3 <!-- Пример xml файла для ЛБ 06 по ООП для ПИН-119 -->
4 <file_1>
5     <student>
6         <family>Иванов</family>
7         <name>Иван</name>
8         <year>1993</year>
9         <group>ПИН-119</group>
10    </student>
11    <student>
12        <family>Петрова</family>
13        <name>Галина</name>
14        <year>1994</year>
15        <group>ПИН-119</group>
16    </student>
17    <student>
18        <family>Борисова</family>
19        <name>Елена</name>
20        <year>1994</year>
21        <group>ПИН-119</group>
22    </student>
23 </file_1>
```

Данный документ состоит из двух основных частей: **пролога** и **корневого документа** (называемого также **элементом документа**). Элемент документа называется здесь FILE\_1, его начальный тег - <FILE\_1>, а конечный - </FILE\_1>, а содержимое - 4 вложенных элемента STUDENT. В свою очередь каждый элемент STUDENT содержит ряд вложенных элементов.

Откройте документ с помощью браузера Internet Explorer. После проверки синтаксиса, документ отобразится на экране. При наличии ошибок вместо документа на экран будет выдано сообщение о невозможности отобразить страницу.



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Имя файла Файл к ЛБ 6.xml -->
<!-- Пример xml файла для ЛБ 06 по ООП для ПИН-119 -->
- <file_1>
-   <student>
-       <family>Иванов</family>
-       <name>Иван</name>
-       <year>1993</year>
-       <group>ПИН-119</group>
-   </student>
-   <student>
-       <family>Петрова</family>
-       <name>Галина</name>
-       <year>1994</year>
-       <group>ПИН-119</group>
-   </student>
-   <student>
-       <family>Борисова</family>
-       <name>Елена</name>
-       <year>1994</year>
-       <group>ПИН-119</group>
-   </student>
</file_1>
```

Создайте XML- документ, представляющий информацию по определенной вариантом предметной области.

### Задание на лабораторную работу

Варианты предметных областей создаваемых XML-документов:

№ вар.	Задача
1	«Человек»: фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира).
2	«Школьник»: фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); школа; класс.
3	«Студент»: фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); ВУЗ; курс; группа; средний бал; специальность.
4	«Покупатель»: фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); номер кредитной карточки; банковского счета.
5	«Пациент»: фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); номер больницы; отделение; номер медицинской карты; диагноз; группа крови.
6	«Владелец автомобиля»: фамилия; имя; отчество; номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира) марка автомобиля; номер автомобиля; номер техпаспорта.
7	«Военнослужащий»:

	фамилия; имя; отчество; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); национальность; дата рождения (год, месяц число); должность; звание.
8	«Рабочий»: фамилия; имя; отчество; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); национальность; дата рождения (год, месяц число); № цеха; табельный номер; образование; год поступления на работу.
9	«Владелец телефона»: фамилия; имя; отчество; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); № телефона.
10	«Абитуриент»: фамилия; имя; отчество; пол; национальность; дата рождения (год, месяц число); домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); оценки по экзаменам; проходной балл.
11	«Государство»: название страны; столица; государственный язык; население; площадь территории; денежная единица; государственный строй; глава государства.
12	«Автомобиль»: марка; цвет; серийный номер; регистрационный номер; год выпуска; год техосмотра; цена.
13	«Товар»: наименование; стоимость; срок хранения; сорт; дата выпуска; срок годности.
14	«Кинолента»: название; режиссер (фамилия; имя); год выхода; страна; стоимость; доход; прибыль.
15	«Рейс»: марка автомобиля; номер автомобиля; пункт назначения; грузоподъемность (в тоннах); стоимость единицы груза; общая стоимость груза.
16	«Книга»: название; автор (фамилия; имя); год выхода; издательство; себестоимость; цена; прибыль.
17	«Здание»: адрес; тип здания; количество этажей; количество квартир; срок эксплуатации; срок до капитального ремонта (25 лет - срок

	эксплуатации).
18	«Программист»: фамилия; имя; отчество; пол; национальность; дата рождения (год, месяц число); образование; номер телефона.
19	«Ученый»: фамилия; имя; отчество; пол; национальность; дата рождения (год, месяц число); ученая степень, должность, номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира).
20	«Пенсионер»: фамилия; имя; отчество; пол; национальность; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира).
21	«Футболист»: фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; название команды; номер в команде; амплуа; результативность (количество голов); количество игр.
22	«Манекенщица»: фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира).
23	«Международная компания»: название; интернет сайт; адрес главного офиса (почтовый индекс, страна, область, район, город, улица, дом, квартира) продолжительность пребывания на мировом рынке; количество сотрудников; количество филиалов в Европе
24	«Телохранитель»: фамилия; имя; отчество; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); дата рождения (год, месяц число).
25	«Зоопарк»: Название животного; количество вида; адрес зоопарка (почтовый индекс, страна, область, район, город, улица, дом, квартира); общее количество животных, количество работников.
26	«Программное обеспечение»: название; название компании производителя; год выхода; цена.
27	«Мультфильм»: название; режиссер (фамилия; имя); год выхода; страна; стоимость;

	доход; прибыль.
28	«Баскетболист»: фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; название команды; номер в команде; амплуа; результативность (количество очков); количество игр.
29	«Область»: название области; областной центр; население; площадь территории; губернатор.
30	«Мотоцикл»: марка; цвет; серийный номер; регистрационный номер; год выпуска; год техосмотра; цена.

## Часть 2. Программное чтение XML файлов

### Теоретическая часть

Списки похожи на массивы, только с более широким спектром управления. В отличие от массивов в список легко добавить новый элемент в конец или вставить элемент в любое место, удалить элемент, сократив список, сортировать простым методом Sort, инвертировать последовательность элементов, добавить к списку другой список.

Списки представляют собой однородную коллекцию значений какого-то типа: чисел, строк или других объектов.

Это может быть список строк файла, список участников мероприятия, список карт в карточной игре на руках у игрока, список слов, загруженных из предложения.

```
List<Buyer> buyers;
List<int> a = new List<int>(); // Создать список из целых чисел
List<double> b = new List<double>(); // Создать список из дробных чисел
List<bool> c = new List<bool>(); // Создать список из булевских значений
"да"/"нет"
List<string> d = new List<string>(); // Создать список строк
List<Card> e = new List<Card>(); // Создать список объектов класса, например,
Card
List<string[]> f = new List<string[]>(); // Создать список массивов
List<List<int>> g = new List<List<int>>(); // Создать список числовых списков
```

### Метод Add

С помощью метода Add можно добавить элементы в список.

```
List<int> x = new List<int>(); //Создали пустой список для хранения
целых чисел
x.Add(5);
```



```
x.Add(27);  
x.Add(-6);  
x.Add(14);  
x.Add(70);  
x.Add(14);  
x.Add(178);
```

Другой же, более сокращённый способ будет выглядеть так:

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };
```

Чтобы получить первое число, используется индекс со значением 0 (счёт начинается с нуля)

```
int m = x[0];    //Значение равно 5  
int n = x[1];    //Значение равно 27
```

А чтобы изменить значение 3-го элемента (индекс 2) на значение 100, используется такая запись:

```
x[2] = 100;
```

### Метод Remove

Метод Remove позволяет удалить из списка элементы с этим значением. Например, попробуем удалить все элементы со значением 14. Будем считать, что у нас добавлены те элементы, что выше

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };  
x.Remove(14);
```

Тогда будут удалены сразу 2 элемента (под номерами 3 и 5). Обратите внимание, что индексы так же идут по порядку. Все элементы, что находились "правее", сдвинулись влево.

### Метод RemoveAt

Данный метод позволяет удалить элемент с определённым индексом

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };  
x.RemoveAt(2);    //Будет удалён 3-ий элемент по счёту
```

### Свойство Count

Считает количество элементов в списке

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };  
int n = x.Count;    //7 элементов  
int m = x.Count - 1; //Получить индекс последнего элемента
```

## Метод Insert

Позволяет вставить новый элемент в определённую позицию. Скажем, мы хотим перед вторым (1-ым по индексу) элементом вставить число 1000. Вставив число, все остальные, что после него, сдвинутся на 1 элемент правее. Пишется это так:

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };
x.Insert(1, 1000);
```

## Метод IndexOf

Данный метод позволяет определить, есть ли в списке элемент с этим значением. Метод IndexOf находит позицию элемента в списке. Если найти не удалось, будет возвращено значение -1.

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };
int a = x.IndexOf(5);      //Получим 0-ую позицию
int b = x.IndexOf(-6);     //Получим 2-ую позицию
int k = x.IndexOf(70);     //Получим 4-ую позицию
int q = x.IndexOf(166);    //Получим -1      }
```

## Другие методы

```
List<int> x = new List<int>() { 5, 27, -6, 14, 70, 14, 178 };
x.Reverse(); //Получить обратный порядок элементов, т.е. 178, 14, 70, 14, -
6, 27, 51
x.Sort(); //Сортировать элементы по порядку с увеличением
int a = x.Min(); //Найти наименьшее значение в списке. Получим -6
int b = x.Max(); //Найти наибольшее значение в списке. Получим 178
int c = x.Sum(); //Найти сумму элементов. Получим 302
double d = x.Average(); //Найти среднее значение чисел. Получим примерно
43,14
```

## Практическая часть

1. Создайте новый проект Visual Studio (шаблон Windows Forms). Разместите на форме компонент DataGridView и две кнопки. Добавьте колонки в DataGridView, согласно вашему заданию (согласно структуре и содержанию файла XML).

В методических указаниях будет разобран файл со следующей структурой:

```
<?xml version="1.0" encoding="utf-8"?>
<buyer>
  <surname>Попов</surname>
  <name>Василий</name>
  <middle_name>Сергеевич</middle_name>
  <gender>мужской</gender>
  <nationality>русский</nationality>
  <height>187</height>
```

```

<weight>60</weight>
<date_of_birth>
  <year>1999</year>
  <month>11</month>
  <number>13</number>
</date_of_birth>
<adress>
  <index>602345</index>
  <country>Россия</country>
  <region>Владимирская</region>
  <area>Новашинский</area>
  <city>Новашино</city>
  <street>Пушкинская</street>
  <house>61</house>
  <apartament>154</apartament>
</adress>
<credit_cart_nomber>6543765404369426</credit_cart_nomber>
<bank_account_number>8375896378264578634785</bank_account_number>
</buyer>

```

Примерное расположение компонент на форме может выглядеть следующим образом:

2. Опишите класс согласно вашему заданию. Стоит отметить, что объявление класса необходимо выполнить до начала класса формы. В рамках примера, объявление класса выглядит следующим образом:

```

public class Buyer
{
    public string Surname { get; set; }
    public string Name { get; set; }
    public string Middlename { get; set; }
    public string Gender { get; set; }
    public string Nationality { get; set; }
}

```

```

    public int Height { get; set; }
    public int Weight { get; set; }
    public DateTime Birthday { get; set; }
    public string Address { get; set; }
    public string CreditCard { get; set; }
    public string AccountNumber { get; set; }
}

```

3. Создайте новое поле (атрибут) формы, в котором будут записаны данные из файла в виде списка классов. Это поле необходимо создать в классе формы, но не в методах!

```
List<Buyer> Buyers = new List<Buyer>();
```

4. Создайте обработчик кнопки «Загрузить». Логика обработчика события:

a. Показываем пользователю диалоговое окно выбора исходного файла.

b. Создаем объект специального класса XmlDocument, который позволит нам выполнить разбор структуры файла.

c. Получаем количество тегов в корневом элементе, которое нам необходимо для создания массива.

d. В цикле обходим все дочерние элементы корневого элемента. Т.е. если в файле есть корневой элемент <buyers> и у него есть дочерние элементы <buyer>, то в пункте «с» - получим количество элементов <buyer>, а в пункте «d» - создадим цикл, позволяющий выполнить обход всех тегов <buyer>.

e. Для каждого тега <buyer> получим все его составные части и запишем в массив структур.

f. Все данные, записанные в массив – выведем в компонент DataGridView.

4a. Создадим диалоговое окно OpenFileDialog. Есть несколько способов: можно перетащить на форму из Панели инструментов, а можно создать программным способом такой объект. Первый способ рассмотрен в ЛБ 9 В этой работе рассмотрим второй способ:

```

OpenFileDialog dialog = new OpenFileDialog();
dialog.Filter = "Файлы xml (*.xml)|*.xml";
if (dialog.ShowDialog() == DialogResult.OK)
{
}

```

Дальше внутри условного оператора необходимо написать остальную логику. Хороший стиль программирования подразумевает отделение интерфейсной части (части, которая обрабатывает события пользовательского интерфейса) от логики приложения, другими словами: писать код, реализующий основные функции приложения непосредственно в обработчиках – плохой стиль программирования. Поэтому вынесем всю оставшуюся логику в отдельные методы: первый метод `private void LoadXMLFile(string path)` – будет содержать логику чтения XML файла в массив, второй - `private void ShowDataInDataGridView()` – будет выводить информацию из массива в `DataGridView`.

Полный код обработчика кнопки «Загрузить»:

```
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog dialog = new OpenFileDialog();
    dialog.Filter = "Файлы xml (*.xml)|*.xml";
    if (dialog.ShowDialog() == DialogResult.OK)
    {
        LoadXMLFile(dialog.FileName);
        ShowDataInDataGridView();
    }
}
```

4b. В новом методе формы `LoadXMLFile` создадим объект типа `XmlDocument`. После создания необходимо указать исходный файл, который будет разобран созданным объектом. И сразу же перейдем к корневому элементу:

```
private void LoadXMLFile(string path)
{
    //Создадим объект для разбора XML файла
    XmlDocument xDoc = new XmlDocument();

    //прочитаем исходный файл
    xDoc.Load(path);

    // получим корневой элемент
    XmlElement xBuyers = xDoc.DocumentElement;
```

4с. XML файл можно представить в виде дерева. Получив корневой элемент можно перейти ко всем дочерним элементам. Другими словами, в переменной `xBuyers` – ссылка на корневой элемент файла. А если у нас есть ссылка на элемент – можно получить его свойства: дочерние элементы, их

количество, атрибуты и внутренний текст. Для начала получить количество дочерних элементов:

```
//Получить количество объектов, описанных в файле
int BuyersCount = xBuyers.ChildNodes.Count;
```

Также для красоты обновим заголовок формы:

```
//установим заголовок формы
this.Text = $"Покупатели. Всего загружено: {BuyersCount}";
```

4d. Запускаем цикл `foreach` для обхода всех дочерних элементов. Так как цикл `foreach` не позволяет получить порядковый номер элемента коллекции, создадим переменную счетчик:

```
// обход всех узлов в корневом элементе
foreach (XmlNode xBuyer in xBuyers)
{
```

4е. Внутри цикла можно получить доступ к содержимому тега `<buyer>` и заполнить массив. Доступ к конкретному дочернему тегу осуществляется оператором индексации (оператор `[]`), в котором нужно указать имя дочернего тега. Для доступа к содержимому тега есть свойство `InnerText` (внутренний текст). Так как файл XML – текстовый все данные тегов мы получаем в типе `string` – поэтому стоит помнить, что для получения числовых данных или даты необходимо выполнить преобразование данных, например, с использованием класса `Convert`.

```
//составные значения
//дата
int Day = Convert.ToInt32(xBuyer["date_of_birth"]
["number"].InnerText);
int Month = Convert.ToInt32(xBuyer["date_of_birth"]
["month"].InnerText);
int Year = Convert.ToInt32(xBuyer["date_of_birth"]["year"].InnerText);

//Адрес
string[] address = new string[8];
address[0] = xBuyer["adress"]["index"].InnerText;
address[1] = xBuyer["adress"]["country"].InnerText;
address[2] = xBuyer["adress"]["region"].InnerText;
address[3] = xBuyer["adress"]["area"].InnerText;
address[4] = xBuyer["adress"]["city"].InnerText;
address[5] = xBuyer["adress"]["street"].InnerText;
address[6] = xBuyer["adress"]["house"].InnerText;
address[7] = xBuyer["adress"]["apartament"].InnerText;
```

```

Buyer buyer = new Buyer()
{
    Surname = xBuyer["surname"].InnerText,
    Name = xBuyer["name"].InnerText,
    Middlename = xBuyer["middle_name"].InnerText,
    Gender = xBuyer["gender"].InnerText,
    Nationality = xBuyer["nationality"].InnerText,
    CreditCard = xBuyer["credit_cart_nomber"].InnerText,
    AccountNumber = xBuyer["bank_account_number"].InnerText,
    Height = Convert.ToInt32(xBuyer["height"].InnerText),
    Weight = Convert.ToInt32(xBuyer["weight"].InnerText),
    Birthday = new DateTime(Year, Month, Day),
    Address = String.Join(", ", address)
};

Buyers.Add(buyer);

```

4f. После заполнения данных в массив необходимо вывести данные на форму в компонент DataGridView. Его внешний вид подготовьте сами: добавьте нужные столбцы, установите их заголовки и ширину. Для заполнения DataGridView мы предусмотрели отдельную функцию ShowDataInDataGridView.

Первым делом нужно очистить данные в DataGridView, на случай если в нем уже были загружены ранее.

```

//удалим все строки из компонента
dataGridView1.Rows.Clear();

```

Затем откроем цикл обхода массива foreach. В цикле добавим строку в коллекцию Rows. Параметрами функции Add являются данные, указанный в том порядке, в котором мы настроили столбцы.

```

//в цикле обойдем массив
foreach (Buyer buyer in Buyers)
{
    //добавим строку, указав нужны данные в порядке следования
    столбцов
    dataGridView1.Rows.Add(buyer.Surname, buyer.Name,
        buyer.Middlename,
        buyer.Gender, buyer.Nationality, buyer.Height, buyer.Weight,
        buyer.Birthday.ToShortDateString(), buyer.Address,
        buyer.CreditCard,
        buyer.AccountNumber);
}

```

### Часть 3. Программная запись XML файлов

## Практическая часть

1. У нас в таблице есть данные – Адрес – эти данные в XML файла разбиты, а в таблице в виде единой строки. Следовательно, при записи данных в XML файл строку с адресом необходимо разбивать на элементы по разделителю «,» (запятая). Однако, если оставить ввод строки в таблицу пользователю – высокая вероятность появления ошибок, скорее всего связанных, с отсутствием части данных адреса. Например: вот так выглядит адрес со всеми заполненными частями:

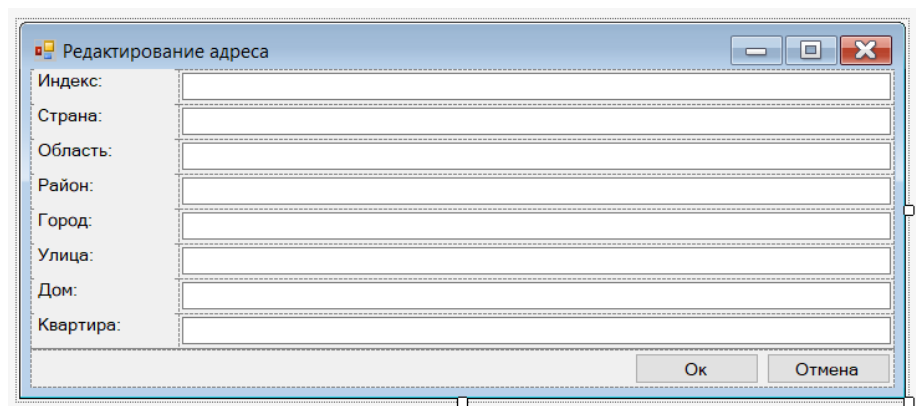
602345, Россия, Владимирская, Новашинский, Новашино, Пушкинская, 61, 154

Но если в адресе часть данных отсутствует, например, район и номер квартиры, адрес должен выглядеть следующим образом:

602345, Россия, Владимирская, , Новашино, Пушкинская, 61,

Но надеяться на пользователя, что он не забудет в этом случае поставить лишние запятые, будет неверным решением. Поэтому, нужно запретить редактирование колонки вручную и добавить функционал защищенного ввода данных об адресе с помощью отдельной формы.

Добавим вторую форму и разместим в ней компоненты в табличном виде (с использованием компонента `TableLayoutPanel`):





Обработчики кнопок устанавливают код возврата диалога и закрывают форму. Также, при нажатии на кнопку Ок выполняется «упаковка» адреса в строку:

```
private void button1_Click(object sender, EventArgs e)
{
    DialogResult = DialogResult.OK;

    string[] temp = new string[8];
    temp[0] = textBox1.Text.Trim();
    temp[1] = textBox2.Text.Trim();
    temp[2] = textBox3.Text.Trim();
    temp[3] = textBox4.Text.Trim();
    temp[4] = textBox5.Text.Trim();
    temp[5] = textBox6.Text.Trim();
    temp[6] = textBox7.Text.Trim();
    temp[7] = textBox8.Text.Trim();

    Address = String.Join(" ", temp);

    Close();
}

private void button2_Click(object sender, EventArgs e)
{
    DialogResult = DialogResult.Cancel;
    Close();
}
```

Теперь при попытке редактирования адреса на форме в таблице необходимо показать форму в модальном режиме и, в случае положительного ответа от пользователя, внести новые данные в таблицу. Событие `CellDoubleClick` возникает при двойном щелчке мышью по ячейке. Следует понимать, что событие срабатывает на любой ячейки таблицы, следовательно, нужно проверить к какой колонке относится ячейка. В нашем случае адрес записывается в колонку с индексом 8. Внутри условного оператора типовой вызов формы в модальном режиме с проверкой на возвращаемый результат. Если пользователь успешно закончил ввод данных, используя входной параметр `e`, получим доступ к индексам строки и столбца ячейки и запишем в свойство `Value` новое значение адреса.

```
private void dataGridView1_CellDoubleClick(object sender,
DataGridViewCellEventArgs e)
```

```

{
    if (e.ColumnIndex == 8)
    {
        FormAddress frm = new FormAddress();
        if(frm.ShowDialog() == DialogResult.OK)
        {
            dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex].Value =
frm.Address;
        }
    }
}

```

## 2. Сохранение данных в XML-файл

Для сохранения данных в XML файл используется класс XmlDocument. Вначале необходимо создать объект типа XmlDocument

```

XmlDocument xDoc = new XmlDocument();

```

Любой XML-документ начинается со строки объявления xml:

```

<?xml version="1.0" encoding="windows-1251"?>

```

Это объявление добавляется в файл специальным методом:

```

xDoc.CreateXmlDeclaration("1.0", "windows-1251", "no");

```

Последовательность наполнения файла:

1. Создание корневого элемента buyers.
2. Создание элемента buyer.
3. Создание элемента с данными (например, name, height)
4. Вставка данных в элемент (внутренний текст)
5. Добавление элемента в качестве дочернего к элементу buyer.
6. Пункты 3-5 повторить для всех тэгов с данными.
7. Добавить заполненный buyer в качестве дочернего к корневому элементу
8. Пункты 2-7 выполнить для все элементов (для каждой строки DataGridView)
9. Заполненный элемент buyer со всеми дочерними элементами сам добавляется дочерним в документ.

Создадим корневой элемент (пункт 1):

```

XmlElement xRoot = xDoc.CreateElement("Buyers");

```

Запустим цикл по всем строкам таблицы:

```
for(int i = 0; i < dataGridView1.Rows.Count - 1; i++)
```

Для удобства текущую строку с которой будем брать данные и записывать их в файл сохраним в отдельную переменную:

```
DataGridViewRow row = dataGridView1.Rows[i];
```

Создадим элемент, который будем наполнять вложенными тегами (пункт 2):

```
XmlElement buyer = xDoc.CreateElement("buyer");
```

Создадим элемент с первыми данными (пункт 3). В нашем случае это surname. Переменную назовем также, как и тег для удобства:

```
XmlElement surname = xDoc.CreateElement("surname");
```

В свойство InnerText внесем данные из строки, обратившись к нулевой колонке (пункт 4):

```
surname.InnerText = row.Cells[0].Value.ToString();
```

Добавим созданный элемент surname дочерним к элементу buyer (пункт 5):

```
buyer.AppendChild(surname);
```

Так сделаем для всех простых данных, не требующих вложенных тегов (пункт 6):

```
XmlElement name = xDoc.CreateElement("name");  
name.InnerText = row.Cells[1].Value.ToString();  
buyer.AppendChild(name);
```

```
XmlElement middle_name = xDoc.CreateElement("middle_name");  
middle_name.InnerText = row.Cells[2].Value.ToString();  
buyer.AppendChild(middle_name);
```

```
XmlElement gender = xDoc.CreateElement("gender");  
gender.InnerText = row.Cells[3].Value.ToString();  
buyer.AppendChild(gender);
```

```

XmlElement nationality = xDoc.CreateElement("nationality");
nationality.InnerText = row.Cells[4].Value.ToString();
buyer.AppendChild(nationality);

XmlElement height = xDoc.CreateElement("height");
height.InnerText = row.Cells[5].Value.ToString();
buyer.AppendChild(height);

XmlElement weight = xDoc.CreateElement("weight");
weight.InnerText = row.Cells[6].Value.ToString();
buyer.AppendChild(weight);

XmlElement credit_card_number = xDoc.CreateElement("credit_card_number");
credit_card_number.InnerText = row.Cells[9].Value.ToString();
buyer.AppendChild(credit_card_number);

XmlElement bank_account_number = xDoc.CreateElement("bank_account_number");
bank_account_number.InnerText = row.Cells[10].Value.ToString();
buyer.AppendChild(bank_account_number);

```

Для составных данных (например, с датой) логика похожая: сначала создаем элемент для даты, потом дочерние элементы, наполняем их данными и добавляем их дочерними:

```

string[] date = row.Cells[7].Value.ToString().Split('.');

XmlElement date_of_birth = xDoc.CreateElement("date_of_birth");

XmlElement number = xDoc.CreateElement("number");
number.InnerText = date[0];
date_of_birth.AppendChild(number);

XmlElement month = xDoc.CreateElement("month");
month.InnerText = date[1];
date_of_birth.AppendChild(month);

XmlElement year = xDoc.CreateElement("year");
year.InnerText = date[2];
date_of_birth.AppendChild(year);

buyer.AppendChild(date_of_birth);

```

Также делаем с адресом:

```

string[] address = row.Cells[8].Value.ToString().Split(',');

XmlElement adress = xDoc.CreateElement("adress");

XmlElement index = xDoc.CreateElement("index");
index.InnerText = address[0];
adress.AppendChild(index);

XmlElement country = xDoc.CreateElement("country");

```

```

country.InnerText = address[1];
adress.AppendChild(country);

XmlElement region = xDoc.CreateElement("region");
region.InnerText = address[2];
adress.AppendChild(region);

XmlElement area = xDoc.CreateElement("area");
area.InnerText = address[3];
adress.AppendChild(area);

XmlElement city = xDoc.CreateElement("city");
city.InnerText = address[4];
adress.AppendChild(city);

XmlElement street = xDoc.CreateElement("street");
street.InnerText = address[5];
adress.AppendChild(street);

XmlElement house = xDoc.CreateElement("house");
house.InnerText = address[6];
adress.AppendChild(house);

XmlElement apartament = xDoc.CreateElement("apartament");
apartament.InnerText = address[7];
adress.AppendChild(apartament);

buyer.AppendChild(adress);

```

Теперь полностью заполненный элемент добавляем дочерним к корневому (пункт 7):

```
xRoot.AppendChild(buyer);
```

После заполнения корневого элемента всеми дочерними добавим сам корневой элемент в xml-документ (пункт 9):

```
xDoc.AppendChild(xRoot);
```

И, последнее что нужно сделать, сохраним документ на диск с помощью метода Save, который в качестве параметра принимает полный путь к файлу (откуда его взять – решать вам: можно, например, вызвать SaveFileDialog):

```
xDoc.Save("Путь к файлу");
```

Здесь стоит отметить один недочет системы: после редактирования данных в DataGridView данные в List не изменяются. Также неправильно

сохранять данные в файл из визуального компонента, правильнее написать метод сохранения List в файл XML.

Это нужно будет сделать самостоятельно.

Также нужно добавить кнопку удаления строки из DataGridView.