№ 5 Windows Forms. Элементы управления

Задание

Создать форму (или формы для ввода агрегируемых объектов). Разместить на ней ЭУ для ввода/вывода информации об объекте (создать свои типы). На форме разместить не менее **9-и различных типов ЭУ** (радиокнопки, списки, поля ввода, метки, кнопки, слайдеры, календарь и т.д.).

Создать дополнительные кнопки для сохранения введенной информации и вывода (отображения сохраненных данных). Запись сохраняемых объектов и чтение выполнять в./из файл типа xml и/или json.

Выполнить валидацию вводимых пользователем данных.

Вариант	Задание
1, 9	**Университет:** Объект – «Студент». Поля: ФИО, возраст, специальность, дата рождения, курс, группа, средний балл, пол, адреса и др. Агрегируемый объект – «Адрес». Поля: город, индекс, улица, дом, квартира. *Дополнительно: Агрегируемый объект – «Место текущей работы». Поля: компания, должность, страж и т.д.*
2, 10	**Банк.** Объект – «Счет». Поля: номер, тип вклада, баланс, дата открытия, владелец, подключение смс оповещения, подключение интернет-банкинга и т.д. Агрегируемый объект – «Владелец». Поля: ФИО, дата рождения, паспортные данные и т.д. *Дополнительно: Агрегируемый объект – «История операций». Поля: тип операции (перевод, снятие…) , сумма, дата и т.д.*
3, 11	**Учебный отдел**. Объект – «Дисциплина». Поля: название, семестр (номер или номера – 1 и 2), курс (на котором читается), специальность (ПОИТ, ДЭВИ, ПОИБМС, ИСиТ), количество лекций в семестре, количество лабораторных, вид контроля (экзамен, зачет), лектор и т.д. Агрегируемый объект: «Лектор». Поля: кафедра, ФИО, аудитория и т.д. *Дополнительно: Агрегируемый объект – «Список литературы». Поля: название, автор, год и т.д.*
4, 12	**Электронная библиотека.** Объект – «Файл книги». Поля: тип (формат), размер файла, название, УДК, количество страниц, издательство, год, список авторов, дата загрузки. Агрегируемый объект – «Автор»: ФИО, страна, ID и т.д. *Дополнительно: Агрегируемый объект – «Издательство». Поля: название, страна, город, год основания, частное или государственное, и т.д.*

Вариант	Задание
5, 13	**Квартира.** Объект «Квартира». Поля: метраж, количество комнат, опции – кухня, ванна, туалет, подвал, балкон и т.р., год постройки, тип материала, этаж и т.д. Агрегируемый объект «Адрес». Поля: страна, город, район, улица, дом, корпус, номер квартиры и т.д. *Дополнительно: Агрегируемый объект – «Комната». Обязательные поля: площадь, количество окон, сторона окон (южная, юго-запад). Комнат может быть несколько.* *Исходя из введенных данных рассчитайте стоимость квартиры (предложите формулу)*
6, 14	**IT лаборатория.** Объект «Компьютер». Поля: тип компьютера (сервер, рабочая станция, ноутбук …..), процессор, видеокарта, размер и тип ОЗУ, размер и тип жесткого диска, дата приобретения и т.д. Агрегируемый объект «Процессор» - поля: производитель, серия, модель, количество ядер процессора, частота, максимальная частота, разрядность архитектуры, размер кэша L1-L3. *Дополнительно: Агрегируемый объект «Видеокарта». Поля: производитель, серия, модель, частота, поддержка DiretX11, объем памяти.* *Исходя из введенных данных рассчитайте стоимость компьютера и всей лаборатории (предложите формулу)*
7, 15	**Аэропорт.** Объект «Самолет». Поля: ID, тип (пассажирский, грузовой, военный), модель (Airbus….) , экипаж (список), количество пассажирских мест, год выпуска, грузоподъемность, дата последнего тех. обслуживания и т.п. Агрегируемый объект «Член Экипажа». Поля: ФИО, должность (пилот, стюардесса…), возраст, стаж и т.д. *Дополнительно: Агрегируемый объект «Производитель». Поля: название, страна, год основания, типы производимых самолетов и т.д.*
8, 16	**Магазин.** Основной объект «Товар». Поля: название, инвентарный номер, размер, вес, тип, дата поступления, количество, цена, производитель. Агрегируемый объект «Производитель». Поля: организация, страна, адрес, телефон. *Дополнительно: Агрегируемый объект «Кладовщик». Поля: ФИО, стаж, адрес и т.д.*

Дополнительные указания

Упрощенный пример

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml.Serialization;
namespace NetW
{
    class Program
    {
        static void Main()
        {
            var role = new List<Role> { new Role { Id = Guid.NewGuid(), Name = "User" }
};
            var users = new List<User>
            {
                new User("Ivan", "Ivanov", 24)
                {
                    Roles = role,
                    Type = UserType.New,
                    Sex = 'M'
                },

                new User("Nikita", "Nikolaev", 24)
                {
                    Roles = role,
                    Type = UserType.New,
                    Sex = 'M'
                },
            };

            XmlSerializeWrapper.Serialize(users, "users.xml");
            var deserializeUsers =
XmlSerializeWrapper.Deserialize<List<User>>("users.xml");

            XmlSerializeWrapper.Serialize(users.First(), "user.xml");
            var deserializeUser = XmlSerializeWrapper.Deserialize<User>("user.xml");
        }
    }

    [Serializable]
    [XmlRoot(Namespace = "NetW")]
    [XmlType("user")]
    public class User
    {
        public User()
        {
            Id = Guid.NewGuid();
        }

        public User(string firstName, string lastName, int age) : this()
        {
            FirstName = firstName;
            LastName = lastName;
            Age = age;
        }

        [XmlIgnore]
        public char Sex { get; set; }

        [XmlElement(ElementName = "id")]
        public Guid Id { get; set; }

        [XmlElement(ElementName = "name")]
        public string FirstName { get; set; }

        [XmlElement(ElementName = "surname")]
```

```
        public string LastName { get; set; }

        [XmlElement(ElementName = "age")]
        public int Age { get; set; }

        [XmlElement(ElementName = "type")]
        public UserType Type { get; set; }

        [XmlArray("roles")]
        [XmlArrayItem("role")]
        public List<Role> Roles { get; set; }
    }

    [Serializable]
    public class Role
    {
        public Guid Id { get; set; }

        public string Name { get; set; }
    }

    [Serializable]
    public enum UserType
    {
        [XmlEnum("L")]
        Locked,
        [XmlEnum("N")]
        New
    }

    public static class XmlSerializeWrapper
    {
        public static void Serialize<T>(T obj, string filename)
        {
            XmlSerializer formatter = new XmlSerializer(typeof(T));

            using (FileStream fs = new FileStream(filename, FileMode.OpenOrCreate))
            {
                formatter.Serialize(fs, obj);
            }
        }

        public static T Deserialize<T>(string filename)
        {
            T obj;
            using (FileStream fs = new FileStream(filename, FileMode.OpenOrCreate))
            {
                XmlSerializer formatter = new XmlSerializer(typeof(T));
                obj = (T)formatter.Deserialize(fs);
            }

            return obj;
        }
    }
}
```

Элементы управления

Ознакомиться с классами ЭУ их методами, свойствами и примерами работы можно на:

https://msdn.microsoft.com/ru-ru/library/3xdhey7w(v=vs.110).aspx

Классы для работы с XML JSON

https://msdn.microsoft.com/ru-ru/library/2bcctyt8(v=vs.110).aspx
https://docs.microsoft.com/ru-ru/dotnet/api/system.runtime.serialization.json?view=netframework-4.7.2

Проверка данных, вводимых пользователем

Практически любое приложение получает данные от пользователя. Если разработчик хочет добиться стабильной работы своего приложения, то следует придерживаться правила: «*Никакая информация введенная пользователем не является полностью надежной и подлежит обязательной проверке*». Проверка может осуществляться двумя способами.

Проверка на уровне поля

Иногда необходимо проверять данные сразу после их ввода. Самым распространенным элементом управления для ввода данных является TextBox. У него в частности есть свойство MaxLength, которое ограничивает число символов, которые можно ввести в текстовое поле.

Элементы управления, принимающие ввод с клавиатуры генерируют следующие события:

KeyDown, KeyUp – любая клавиша нажата, любая клавиша отпущена. Когда пользователь нажимает клавишу, которой соответствует значение ASCII, генерируется событие KeyPress. Именно последнее событие применяется для проверки «на лету» вводимых символов. При генерации события обработчик получает экземпляр класса KeyPressEventArgs. У него есть два свойства: KeyChar – возвращает нажатый символ. Значение этого свойства можно проверять с помощью статических методов класса Char на принадлежность к той или иной группе.

Char.IsDigit, Char.IsLetter, Char.IsLetterOrDigit, Char.IsPunctuation, Char.IsLower, Char.IsUpper.

Второе свойство – Handled. Определяет, было ли событие обработано. Если оно принимает значение false, то событие считается необработанным и пересылается операционной системе для дальнейшей обработки, если true – то событие считается обработанным и дальнейших действий не происходит. Например, мы хотим организовать текстовое поле, куда можно вводить только цифры.

```
private void myTextBox_KeyPress(object sender, KeyPressEventArgs e)
{
    e.Handled = !Char.IsDigit(e.KeyChar);
}
```

Здесь обязательно отрицание. Однако, может получиться ситуация, когда каждый отдельный символ является правильно введенным, но весь текст является некорректным. Например, введенный текст является числом, но оно выходит за допустимые логикой программы пределы.

Для такой проверки используется обработка события Validating, которое генерируется ПЕРЕД потерей фокуса элементом управления. Событие будет

сгенерировано только в том случае, если свойство CausesValidation у данного элемента и у элемента получающего фокус следующим установлено в true. В обработчик события передается объект класса System.ComponentModel.CancelEventArgs. У этого класса есть свойство Cancel, если его установить в true, то дальнейшая обработка будет приостановлена и фокус вернется к исходному элементу управления.

После успешной проверки значения элемента сгенерируется событие Validated.

Проверка на уровне формы

Она позволяет проверить одновременно все поля формы. Например, в двух текстовых полях задаются границы интервала, каждая из них может быть нормальным числом, не выходящим за допустимые пределы, но при этом как границы они будут неверны.

Для запуска проверки обычно используется событие Closing формы или событие нажатия кнопки (OK, Применить и т.д.). В обработчик события Closing также передается объект класса CancelEventArgs.

Оповещение пользователя об ошибках ввода

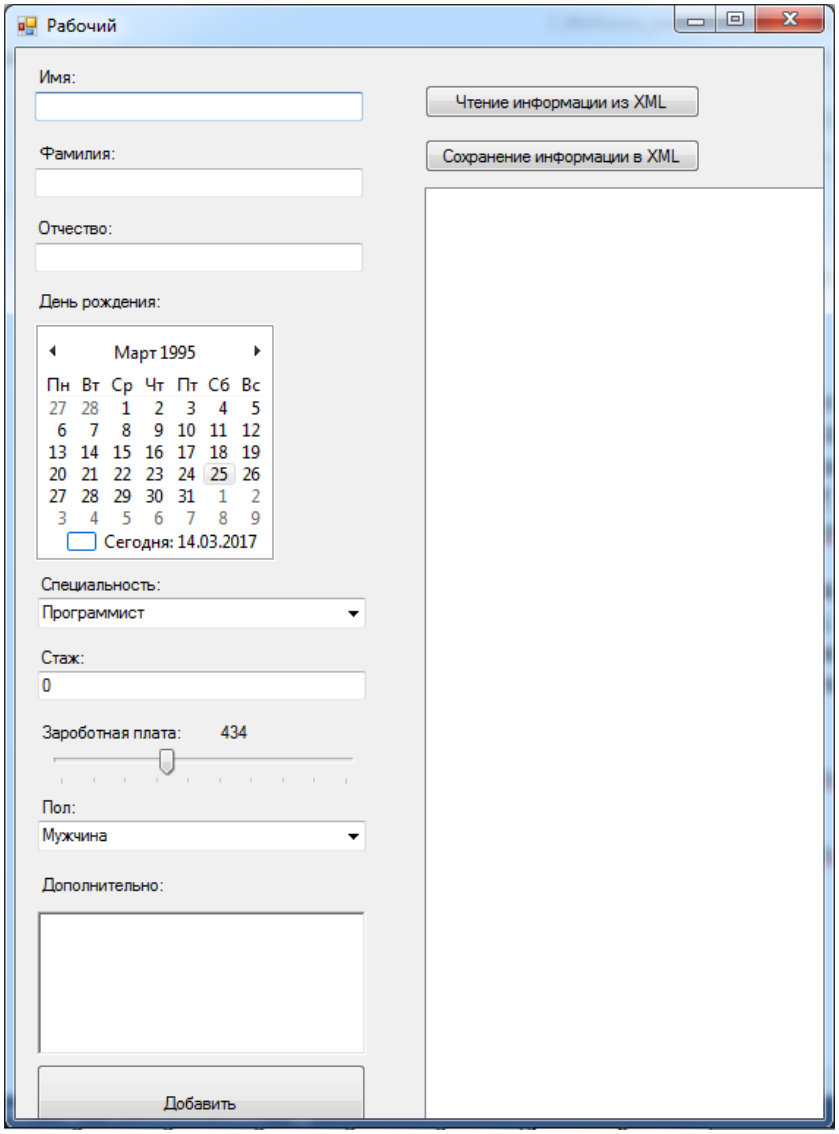
Делать это можно по разному:

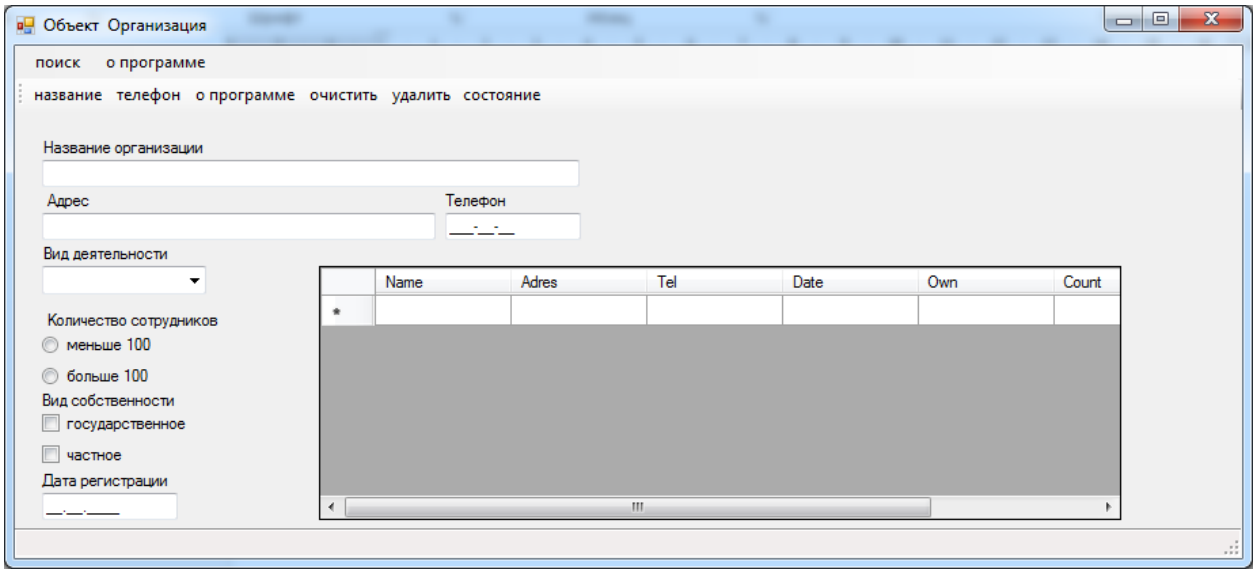
- с помощью модального окна с сообщением MessageBox.Show("Ошибка ввода");
- с помощью изменения цвета текста или фона текстового поля с неверно введенными данными;
- с помощью компонента ErrorProvider

Компонент позволяет задать для каждого элемента управления текст сообщения об ошибке, если этот текст задан, то рядом с элементом управления появляется мигающий значок (его вид можно задать в свойствах компонента). А при наведение указателя мыши на значок появляется текст сообщения об ошибке.

```
if(notEmptyTextBox.Text.Equals(""))
      myErrorProvider.SetError(notEmptyTextBox, "Поле не может быть пустым");
else
      myErrorProvider.SetError(notEmptyTextBox, "");
```

Примеры интерфейса:





Вопросы:

1. Какое основное назначение технологии Windows Forms, ее особенности, преимущества и недостатки?
2. Зачем используется класс Form? Назовите основные методы, свойства и события данного класса.
3. Поясните структуру проекта и назначение всех файлов?
4. Зачем нужен атрибут STAThreadAttribute?
5. Как в вашем проекте используются события и делегаты?
6. Объясните схему работы цепочек делегатов.
7. Объясните механизм подписки и отмены подписки на события.
8. Как создать вторую форму и передать туда данные? Есть ли другие способы?
9. Как во время выполнения приложения добавить/удалить контрол?