# Практическая работа №13

**Тема: Разработка приложений с несколькими формами**

**Цель работы:** формирование навыков создания приложений Windows Forms с использованием меню и диалоговых окон

# Задачи:

* изучение приемов создания приложений Windows Forms;
* создание проектов с использованием меню и диалоговых окон.

# Материально-техническое обеспечение:

**Место проведения:** Компьютерный класс.

# Время на выполнение работы: 2 часа.

**Оборудование**: ПК

**Средства обучения:** операционная система, текстовый процессор MS Word, программные средства определенного вида

# Исходные данные:

1. Конспект занятия.
2. Задание для практической работы.

# Перечень справочной литературы:

1) Программирование на языке высокого уровня. Программирование на языке С++: учеб. пособие / Т. И. Немцова, С. Ю. Голова, А. И. Терентьев ; под ред. Л. Г. Гагариной. – М. : ИД

«ФОРУМ» : ИНФРА-М, 2018. – 512 с. – (Среднее профессиональное образование).

# Краткие теоретические сведения:

**Использование меню и диалоговых окон**

Библиотека .NET содержит компоненты позволяющие проектировать приложения с развитым графическим интерфейсом, включающим меню, панели инструментов, разнообразные диалоговые окна. Некоторые из них сразу отображаются на форме, другие могут вызываться.

## Основные компоненты для построения меню:

**ToolStrip** – панель инструментов (базовый класс – контейнер). На ней размещают элементы – объекты **ToolStripItem**.

Наследник **ToolStrip** – полоска меню **MenuStrip**. **StatusStrip** – строка (полоска) состояния.

Основные свойства элементов меню: вид, размеры, позиционирование (**Dock**, **LayoutStyle**), видимость (ShowItemToolTips). Основные события связаны с выбором конкретного пункта меню щелчком мыши и имеют синтаксис: пункт **ToolStripMenuItem\_Click.**

Важнейшей особенностью Windows-приложений является оконный интерфейс, включающий

окна разного типа, имеющие свое назначение, функционал и внешний вид. Каждое приложение имеет одно главное окно. Напомним, что класс главного окна приложения содержит точку входа в приложение (статический метод **Main**). При закрытии главного окна приложение завершается. **Модальные и немодальные формы**

В графическом интерфейсе операционной системы Windows различают два вида диалоговых окон:

* модальные окна;
* немодальные окна.

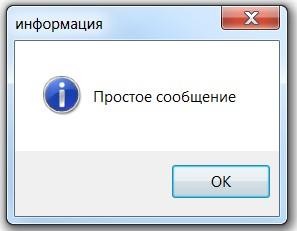
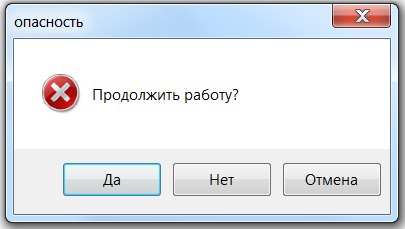
*Модальные окна* являются наиболее распространенным видом окон. Если открыто модальное окно, то чтобы попасть в основную программу, его обязательно нужно закрыть. Без закрытия модального окна невозможно перейти в программу, которая его вызвала. В программах Windows Forms возможна ситуация, когда одно модальное окно открывает другое (следующее) модальное окно. Таким образом получается цепочка вызовов.

Как правило, модальное окно содержит кнопки типа **OK, Cancel** (или другие), при нажатии на которые это окно закрывается.

**Примером** модального окна может быть команда Open… редактора Microsoft Word, которая открывает окно выбора файла. Пока не будет выбран файл или не будет нажата одна из кнопок отмены, перейти в редактор не удастся.

При открытии *немодального окна*, существует возможность переключаться из окна в основную программу не закрывая это окно, и, наоборот, из основной программы можно снова вернуться в немодальное окно. Но при этом, визуально немодальное окно всегда размещается над основной программой. Как правило, немодальное окно содержит одну кнопку типа Close, которая его закрывает.

**Пример** немодального окна – это команда «Insert Symbol» редактора Microsoft Word, которая позволяет вставлять символ в документ. Из окна вставки символа можно переключаться в основной документ и наоборот, из документа можно переходить в немодальное окно.

В виде модальных обычно оформляют диалоговые окна, требующие от пользователя ввода какой-либо информации или подтверждения. Наиболее часто используют **Окно сообщений (класс MessageBox)**, которое вызывается методом **Show()** и может принимать ряд параметров (рис 1): **text** – текст выводимого сообщения, **caption** – текст заголовка окна сообщения, **icon** – значок окна сообщения, **buttons** – используемые кнопки.

*Рис. 1. Вид диалогового окна сообщений MessageBox*

Это окно может иметь от 1 до 3 кнопок (рис.1). Количество и назначение задается значением свойства **MessageBoxButtons**: **OK** – одна кнопка **OK**; **OKCancel** – две кнопки (**OK** и **Отмена**); **YesNo** – две кнопки (**Да** и **Нет**); **YesNoCancel** – три кнопки (**Да**, **Нет**, **Отмена**). Метод **Show()** возвращает объект **DialogResult**, позволяющий узнать, какая кнопка в окне была нажата. Вид значков в соответствии с содержанием сообщения можно задавать значением свойства **MessageBoxIcon**: **Information** – буква i в кружке; **Error** – белый знак «**X**» в красном круге; **Warning** – восклицательный знак в желтом треугольнике; **Question** – вопросительный знак в круге.

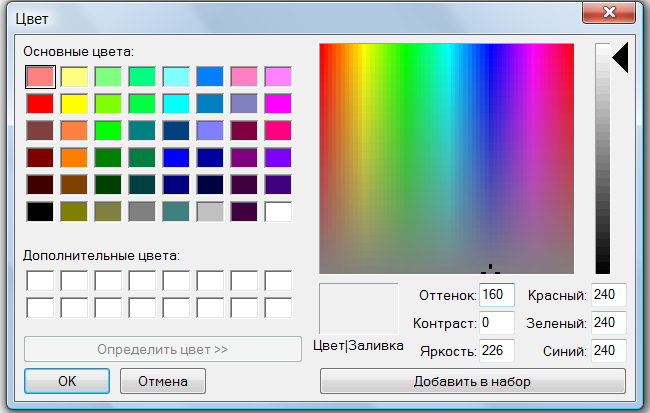
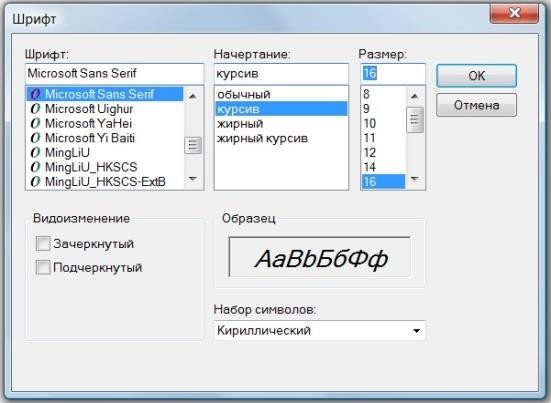
Выполнение многошаговых операций значительно упрощается благодаря использованию специализированных диалоговых окон (или просто диалогов), имеющих набор требуемых свойств, методов и событий. Эти окна отображаются методом **ShowDialog().** Приведем примеры таких диалогов.

**ColorDialog** – выбор цвета (рис.2, а). Возвращает выбранный цвет:

## colorDialog1.ShowDialog(); this.BackColor = colorDialog1.Color;

**FontDialog** – выбор шрифта (рис.2, б). Возвращает выбранный шрифт:

## fontDialog1.ShowDialog(); textBox1.Font = fontDialog1.Font;

*Рис.2. Вид диалоговых окон ColorDialog (а) и FontDialog (б)*

Диалоги **OpenFileDialog** (открытие файла) и **SaveFileDialog** (сохранение файла) возвращают дескрипторы выбранного файла, например:

***string fn = openFileDialog1.FileName; или string fn = saveFileDialog1.FileName;***

# Особенности работы с немодальными окнами в Windows Forms

В технологии Windows Forms основной программе соответствует форма – экземпляр класса, унаследованного от класса Form. Также модальному или немодальному окну соответствует некоторая форма.

При работе с немодальными окнами были замечены следующие особенности:

* в немодальном окне есть одна кнопка Close, которая его закрывает;
* вызов немодального окна из главной программы происходит с помощью метода Show() класса Form (чтобы вызвать модальное окно нужно использовать метод ShowDialog()).

В нижеследующем коде приведены примеры вызова модального и немодального окна

// 1. Вызов модального окна

**dlgResult = modalForm.ShowDialog();**

// Обработка результата возврата, который размещается в dlgResult

// ...

// 2. Вызов немодального окна

**nonModalForm.Show();** // здесь не нужно ничего обрабатывать

Как видно из вышеприведенного кода, проблемой есть способ обмена информацией между основной программой и немодальным окном. Основной программе соответствует главная форма, немодальному окну соответствует немодальная форма.

# Способы обмена информацией между главной формой и немодальной формой

При использовании немодальных форм важным есть обмен информацией между основной программой (главной формой) и немодальной формой.

Здесь возможны две ситуации:

* передача информации из главной формы в немодальную. Эта ситуация решается без проблем, поскольку в основной форме создается экземпляр немодальной формы. Через этот экземпляр осуществляется доступ к компонентам немодальной формы стандартным способом;
* передача информации из немодальной формы в главную форму. Здесь возникает проблема: как из немодальной формы осуществить доступ к экземпляру главной формы.

Чтобы передать данные (информацию) с немодальной формы в главную, можно использовать один из следующих способов:

1. *С помощью статической ссылки* на главную форму. Как известно, статические переменные есть глобальными, поэтому удобно могут использоваться для обмена информацией между разными разделенными экземплярами или процессами. Имея ссылку на главную форму, можно легко обращаться к компонентам этой формы. Этот способ наиболее прост в использовании.
2. *С помощью механизма событий*. При этом способе в немодальной форме объявляется событие, которое возникает каждый раз, когда главной форме нужно передать информацию. Тем не менее, сам обработчик этого события помещается в главной форме. Если из немодальной формы нужно передать информацию главной форме, то возникает событие. В результате, в главной форме вызывается обработчик этого события. Этот обработчик получает объект типа

«немодальное диалоговое окно», который сгенерировал событие. Имея доступ к измененному объекту немодального окна, можно обновить информацию в главной форме.

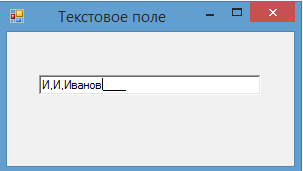
1. *С помощью записи информации во временный файл*. Как известно, файлы подобны статическим переменным. Поэтому, записав порцию информации в файл в немодальной форме, можно ее прочитать в главной форме. Этот способ не является эффективным, поскольку работает медленнее (чтение из носителей информации происходит дольше чем из оперативной памяти). **Элемент MaskedTextBox -**

по сути представляет обычное текстовое поле. Однако данный элемент позволяет контролировать ввод пользователя и проверять его автоматически на наличие ошибок

Чтобы контролировать вводимые в поле символы, надо задать маску. Для задания маски можно применять следующие символы:

* + 0: Позволяет вводить только цифры
  + 9: Позволяет вводить цифры и пробелы
  + #: Позволяет вводить цифры, пробелы и знаки '+' и '-'
  + L: Позволяет вводить только буквенные символы
  + ?: Позволяет вводить дополнительные необязательные буквенные символы
  + A: Позволяет вводить буквенные и цифровые символы
  + .: Задает позицию разделителя целой и дробной части
  + ,: Используется для разделения разрядов в целой части числа
  + :: Используется в временных промежутках - разделяет часы, минуты и секунды
  + /: Используется для разделения дат
  + $: Используется в качестве символа валюты

Чтобы задать маску, надо установить свойство Mask элемента.

Например, можно создать свою маску для ввода инициалов имени и отчества и фамилий ограниченной длины (до 15) в текстовое поле. Для этого присвоим

свойству **Mask** значение **L.L.L??????????????**. Тогда ввод в текстовое поле будет выглядеть следующим образом:

Данный элемент также представляет нам ряд свойств, которые можно использовать для управления вводом. Так, свойство **BeepOnError** при установке значения true подает звуковой сигнал при введении некорректного символа.

Свойство **HidePromptOnLeave** при установке в true при потери текстовым полем фокуса скрывает, указанные в PromptChar

Свойство **PromptChar** указывает на символ, который отображается в поле на месте ввода символов. По умолчанию стоит знак подчеркивания.

Свойство **AsciiOnly** при значении true позволяет вводить только asci-символы, то есть символы из диапазона A-Z и a-z.

**Ход работы:**

# Требования к содержанию отчета:

* + Номер и название практической работы.
  + Цель работы.
  + По каждой заданию (задаче/примеру) экранные формы (при наличии) и листинг программного кода, показывающие порядок выполнения практической работы, и результаты, полученные в ходе её выполнения.
  + Ответы на контрольные вопросы в тетради.

# Порядок выполнения работы:

Все проекты практической работы размещать в своей сетевой в новой папке **Пр14\_ФИО**

# В начале каждого файла проекта установить комментарии: пр.р.№ (указать номер), свою Фамилию. Формулировку задания

**Задание 1.** Создание приложения с использованием меню.

# Создание простейшего текстового редактора. Шрифт и цвет текста изменяется с помощью меню.

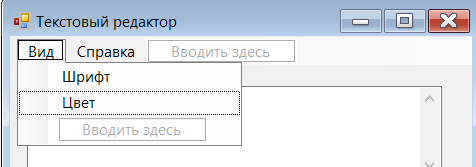
1. Запустите **MS Visual Studio2019.** Создайте новый проект **Приложение Windows Forms (.NET Framework)** для **C#**, имя проекта **pr13\_Прил1\_Фамилия.**
2. Создайте форму, в соответствии с рис.1.1

Разместите на форме текстовое поле **textBox1** размером **480x360** (свойства: **Multiline = true, ScrollBars = Vertical, Anchor = Top, Bottom, Left, Right**).

Перетащите на форму компоненты из группы *Диалоговые окна*:

**ColorDialog**, **FontDialog** (на форме они не отображаются) и из группы *Меню и панели инструментов* **MenuStrip**.

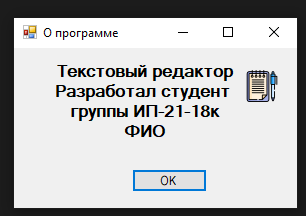
1. Выделите компонент **MenuStrip**. Пользуясь подсказками, создадим два пункта меню: вид (с подпунктами шрифт, цвет) и справка (рис 2).



*Рис.2 Создание меню*

1. Поочередно выделяйте созданные пункты и регистрируйте события **Click**.
2. В шаблоны обработчиков выбора пунктов меню **шрифт** и **цвет** введите коды, вызывающие диалоги задания шрифта и цвета текста:
3. Изображение выглядит как текст

   Автоматически созданное описаниеВ В шаблон обработчика выбора пункта меню **Справка** будет открываться новая форма. Добавим форму в проект. По нажатию на кнопку ОК, форма должна закрываться.

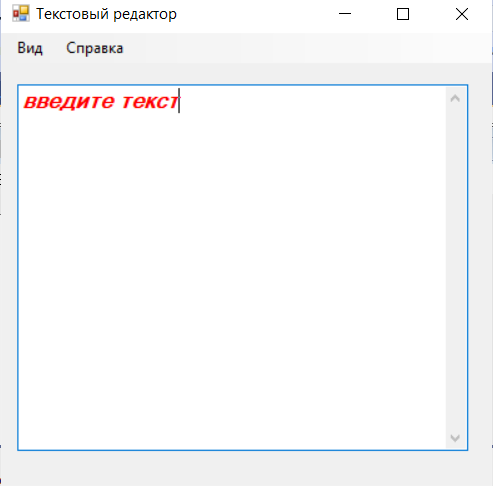
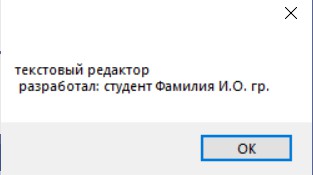


7. Добавим обработчик события

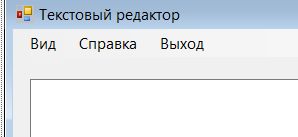
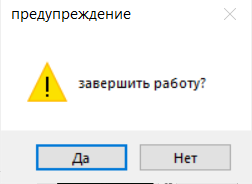
Изображение выглядит как текст

Автоматически созданное описание

8. Протестируйте программу, вводя текст и изменяя цвет и шрифт. Результат может выглядеть так (рис. 1.2, а, б).

9. Добавьте пункт меню **Выход**. Зарегистрируем событие **Click**. В шаблон обработчика введите код вызова окна сообщений с двумя кнопками (**Да – Нет**) и иконкой предупреждения (рис.1.3.):



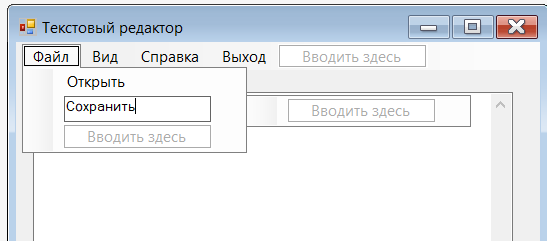
1. Протестируйте окончательный вариант.

**Задание 2.** Измените приложение текстового редактора, добавив возможность открывать и сохранять текстовые файлы.

# Пример 2. Открытие и сохранение текстовых файлов.

1. Для этого в начале файла с программным кодом **Form1.cs** подключите пространство имен

# System.IO.

1. Перетащите на форму компоненты: **ОpenFileDialog** (открытие) и **SaveFileDialog** (сохранение файла).
2. Добавьте в меню пункт **Файл** с подпунктами **Открыть** и **Сохранить** и зарегистрируйте для них события **Click**.
3. В шаблоны обработчиков введите коды вызова диалогов открытия и сохранения файлов:



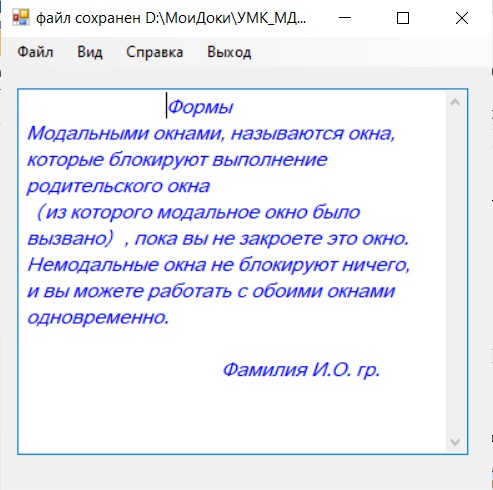
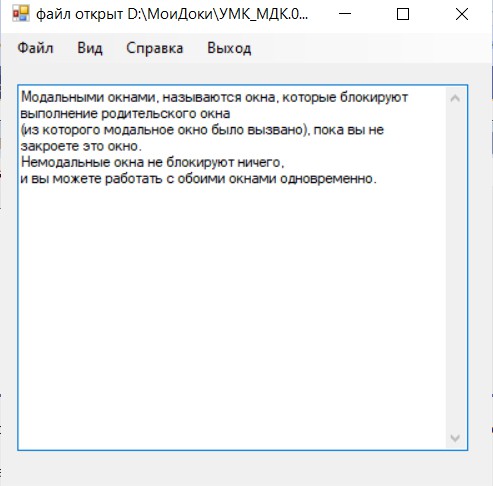
1. Протестируйте программу. Откройте текст из файла ***текст1.txt*** (рис. 1.4.а). Обратите внимание на заголовки окон при выполнении операций

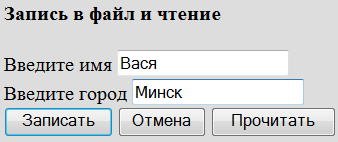
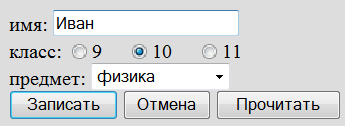
Рис. 1.4. Фрагмент текста до (а) и после редактирования (б)

1. Отредактируйте и оформите текст: вставьте заголовок «*Формы*», а в конце свою *фамилию* и *группу*; шрифт *Arial, 12 пт*, *курсив*, *цвет синий* (рис. 3.22, б). Сохраните файл под именем *текст1 с изм.txt.*

# Задания для самостоятельной работы

1. Создайте приложение, в котором двойным щелчком мыши на форме вызывается компонент **colorDialog**, позволяющий изменять цвет формы.
2. Создайте приложение, в котором по нажатию кнопки вызывается компонент

**openFileDialog**, позволяющий выбирать и загружать в **pictureBox** изображение из файла.

1. Создайте регистрационную форму, в которую вводится имя и город. Данные можно записывать в файл **sam3.txt** и читать по нажатию кнопок.
2. Создайте регистрационную форму, в которой вводится имя, с помощью radioButton выбирается класс, из списка **comboBox** выбирается предмет (физика, математика,

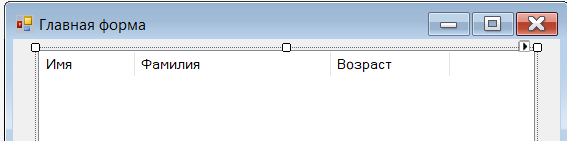
информатика). Данные можно записывать в файл **reg.txt** и читать из файла по нажатию кнопок.

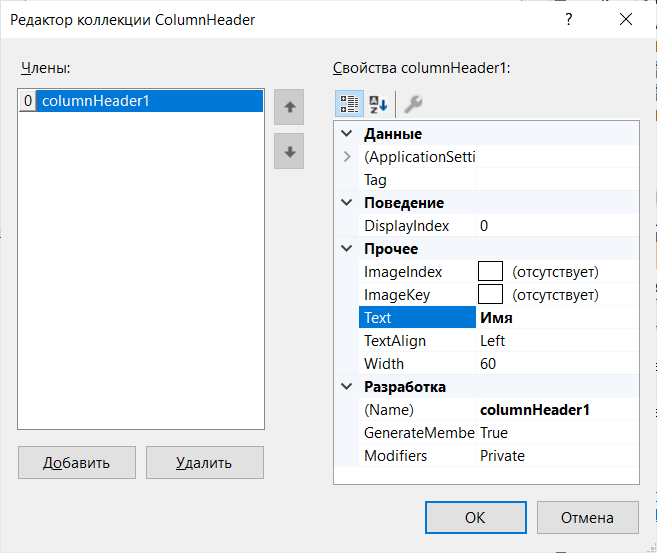
**Задание 3.** Создайте приложение с использованием нескольких форм.

# Пример 3. Диалоговые окна

Классическая задача — получение ввода от пользователя — чаще всего решается через диалоговые окна. Например, у вас есть форма для отображения списка людей, и вы хотите иметь возможность добавления людей в список и редактирования их. Где расположить поля ввода для параметров добавляемого или редактируемого объекта? Можно в том же окне, где находится список, но лучше это делать в отдельном окне редактирования. Давайте рассмотрим это на примере.

1. Создайте новый проект **Приложение Windows Forms (.NET Framework)** для **C#**, имя проекта **pr14\_Прил2\_Фамилия.**, На главной форме которого поместите компонент **ListView** для хранения списка и **две кнопки** для добавления новой записи (свойство *Name* **addPersonButton**) и редактирования выделенной (свойство *Name* **editPersonButton**).

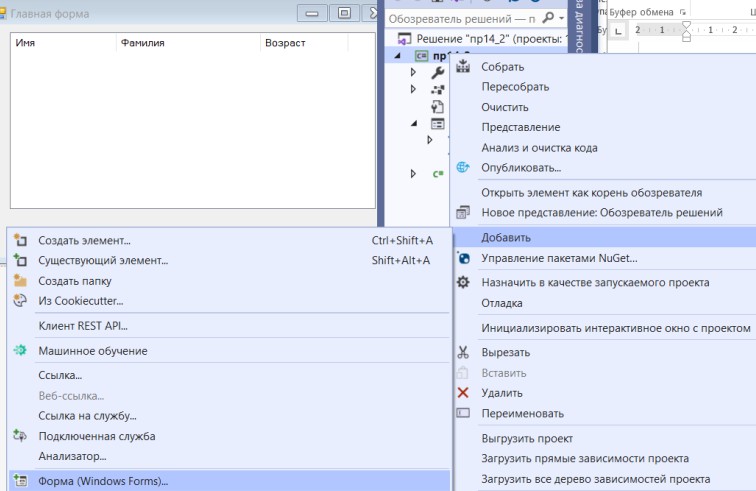


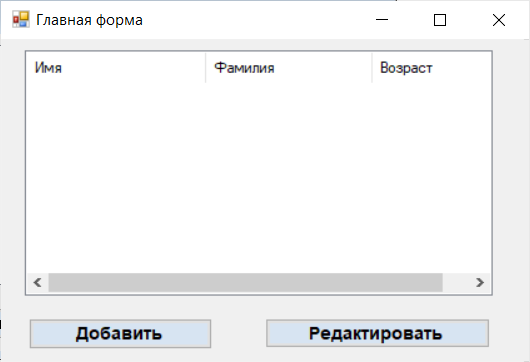
1. В списке **ListView** нужно:
   * в свойстве *Name* задать **personsListView**
   * в свойстве **Columns** добавить три колонки с заголовками: **Имя**, **Фамилия** и **Возраст**;
   * в свойстве **View** выбрать **Details**.

Такую главную форму можно увидеть на рис. 2.1.

1. Теперь нам понадобится дочерняя форма или диалоговое окно для редактирования элементов. Щелкните правой кнопкой мыши по имени проекта (**pr14\_2…**) в панели **Обозреватель решений**

(Solution Explorer) и в контекстном меню выберите **Добавить** (Add) - **Форма (Windows Form)**.

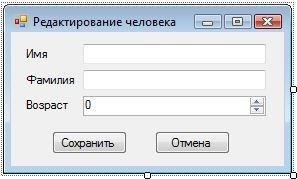
1. В открывшемся окне найдите и выделите **Форма (Windows Form)** и в поле **Name** введите имя для диалогового окна — например, **EditPersonForm**.



*Рис.2.1 Форма будущей программы*

1. Поместите на форму **две кнопки** для *сохранения* и *отмены* изменений, а также **три поля ввода:** для *изменения имени* (свойство *Name* **firstNameTextBox**)*, фамилии* (свойство *Name* **lastNameTextBox**) *и возраста*. Для изменения возраста можно использовать компонент **NumericUpDown** (свойство *Name* **ageNumericUpDown**), который хорошо подходит для редактирования чисел.

Примерный вариант формы на рис. 2.3:



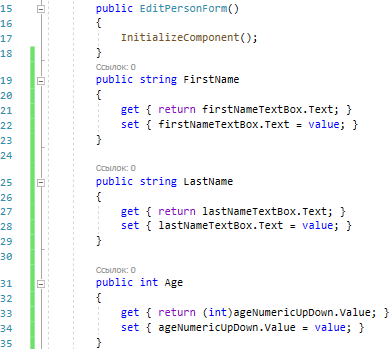
*Рис. 2.3. Форма диалогового окна*

1. Выполните некоторые рекомендации по оформлению диалоговых окон, которым не обязательно следовать, но которые чаще всего положительно сказываются на удобстве использования программ:
   * в свойстве **AcceptButton** нужно указать кнопку, которая будет являться кнопкой по умолчанию, и именно она станет срабатывать (будет вызываться ее обработчик события **Click**), если пользователь нажмет клавишу в тот момент, когда отображается окно. Здесь чаще выбирают кнопки типа **OK** или **Да**;
   * в свойстве **CancelButton** укажите кнопку отмены, которая будет срабатывать при нажатии клавиши ;
   * свойство **FormBorderStyle** лучше установить в **FixedDialog**, потому что диалоговым окнам желательно устанавливать фиксированный размер окна, их просто незачем растягивать;
   * свойство **MaximizeBox** рекомендуется установить в **false**, чтобы в заголовке окна не отображалась кнопка максимизации формы. Мы уже определились, что размер окна не должен изменяться, и возможность максимизации тем более не нужна;
   * свойство **ShowInTaskBar** лучше тоже установить в **false**, потому что только главное окно должно отображаться в панели задач. Дочерние окна отображайте в панели задач только при необходимости.
2. Далее выполните пару манипуляций над кнопками:

Для кнопки **Сохранить** установите свойство **DialogResult** в **OK**, а для кнопки **Отмена** установите это же свойство в **Cancel**. Теперь, если окно отображается в модальном режиме, то по нажатию любой из этих кнопок окно будет закрыто, а форма вернет в качестве результата то, что мы указали для кнопок в свойстве **DialogResult**.

1. Теперь посмотрим, как может использоваться эта форма.

Для начала в коде формы добавим три свойства, через которые будет происходить обращение к полям ввода. Перейдите в окно кода формы (**ПКМ** по файлу **EditPersonForm.cs** – **Перейти к коду**):

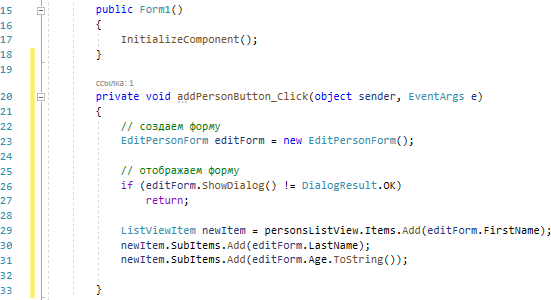


Добавленные здесь свойства являются посредниками между диалоговым окном редактирования и главным окном, из которого будет вызываться это диалоговое окно.

Теперь все готово, чтобы написать код, который будет создавать новые элементы в списке и редактировать их.

1. Для начала создайте обработчик события **Click** для кнопки **Добавить** главной формы и напишите в нем содержимое листинга 1.

# Листинг 1. Код добавления нового элемента в список

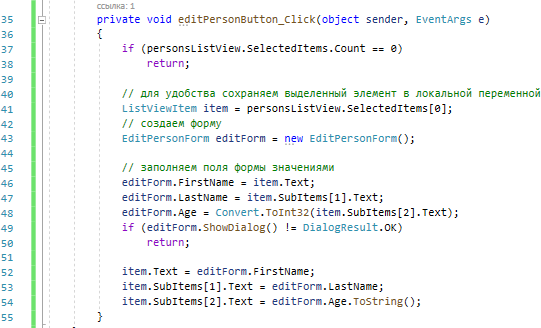


В самом начале создается экземпляр класса **EditPersonForm**, а этим классом является форма. Следующим этапом идет отображение формы **editForm.ShowDialog()**, и тут же проверяется результат, который возвращает метод **ShowDialog().** Если результат не равен **DialogResult.OK** (а этот результат возвращает кнопка **Сохранить** в диалоговом окне), то выполнение метода прерывается.

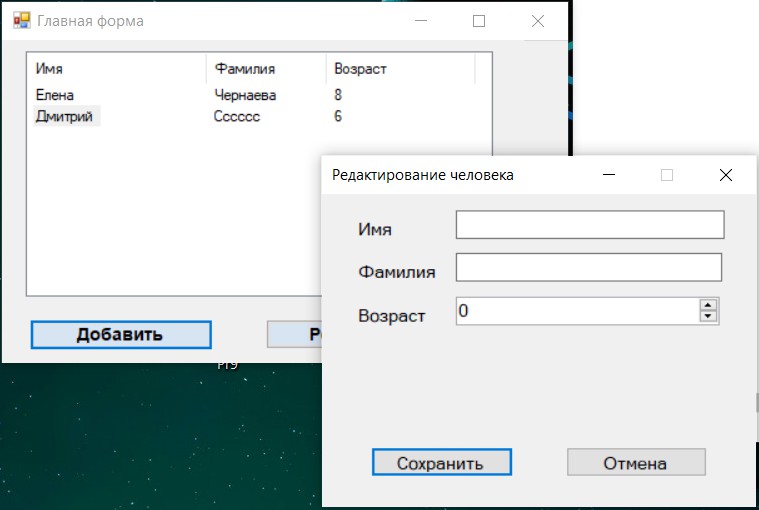
После этого создается новый элемент для списка **ListView** и заполняются два подэлемента, которые будут отображаться в колонках. Для чтения данных из объекта диалогового окна **editForm** используются свойства, которые мы написали.

1. Теперь посмотрим на метод, который выполняет редактирование выделенного в списке элемента. Напишите код в листинге 2.

# Листинг 2. Редактирование с помощью диалогового окна



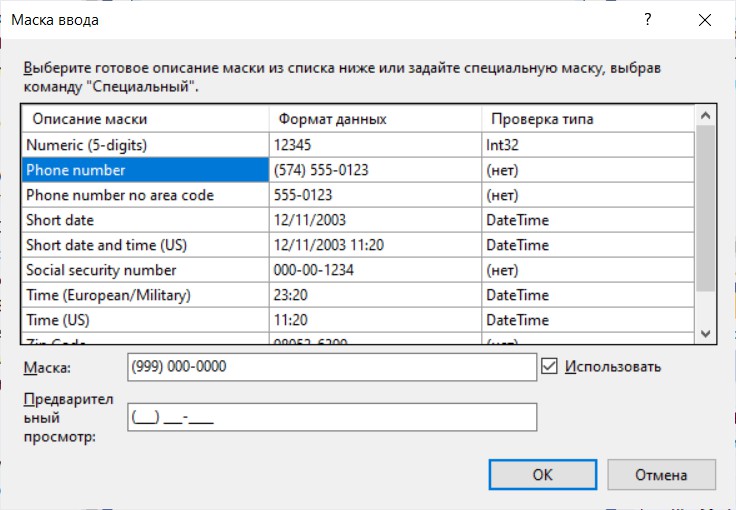
1. Протестируйте работу программы. При сохранении данных, они должны добавляться в список главной формы. Выделив нужную запись списка при нажатии кнопки **Редактировать** должно открываться окно редактирования.



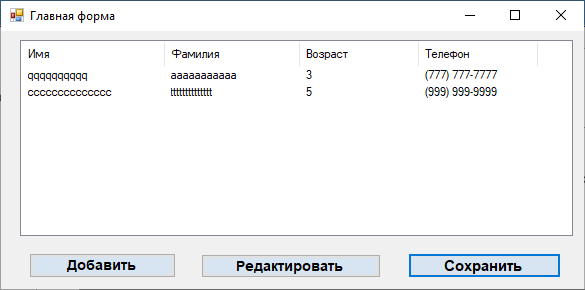
Обратите внимание, что если закрыть главную форму, то закрывается и дочерняя форма.

1. Добавьте еще одно поле **Телефон** на Главную форму, как дополнительный столбец и на дочернюю форму: для этого понадобится **label** и элемент **MaskedTextBox**, который по сути представляет обычное текстовое поле, но позволяет контролировать ввод пользователя и проверять его автоматически на наличие ошибок.

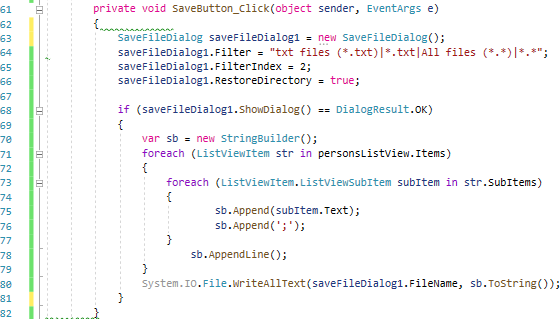
Чтобы задать маску, надо установить свойство **Mask** элемента. Найдя это свойство в окне свойств (**Porperties**), нажмите на него и вам отобразится окно для задания одного из стандартных шаблонов маски. В частности мы можем выбрать **Phone number (Телефонный номер)**, который подразумевает ввод в текстовое поле только телефонного номера:



Теперь при запуске мы сможем ввести в текстовое поле только цифры, получив в итоге телефонный номер.

1. Добавьте на главную форму кнопку **Сохранить** и необходимый для нее компонент, а также свойства.

Напишите программный код сохранения данных из таблицы в файл:



**Задание для самостоятельной работы.** Создайте главную форму, отображающую данные из регистрационной формы, в которой в текстовые поля вводят *имя, логин, пароль и e-mail*. Корректность ввода проверяется с помощью регулярных выражений. Если введены верные данные, они отображаются в главной форме и сохраняются в файле **log.txt**, иначе выводится сообщение «**Повторите ввод**».

# Контрольные вопросы:

1. Опишите основные компоненты для построения меню
2. Модальные и немодальные формы
3. Особенности работы с немодальными окнами в Windows Forms
4. Способы обмена информацией между главной формой и немодальной формой
5. Окно сообщений: описание, свойства
6. Элемент MaskedTextBox: назначение, свойства