ПРАКТИЧЕСКАЯ РАБОТА № 16

(4 академических часа)

Тема: Создание Windows-приложений. Создание тестового редактора.

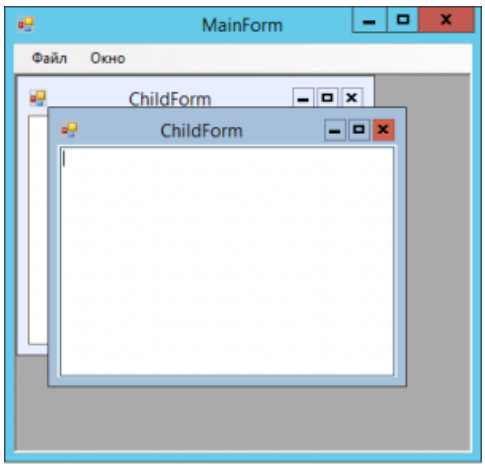
Цель задания:

1. Понять основы проектировании приложений с многооконным пользовательским интерфейсом.
2. Научиться работать с невизуальными компонентами: меню, окна диалога и т.п.
3. Научиться реализовывать операции обмена данных через универсальный буфер обмена Clipboard.

Теоретическая часть.

В приложениях MDI имеется родительская (первичная) форма и ряд однородных дочерних форм (называемых также формами документов). Число дочерних окон заранее неизвестно – пользователь может создать их столько, сколько ему потребуется. Окна документов располагаются в клиентской области родительской формы. Поэтому целесообразно в родительской форме ограничиваться только главным меню, инструментальными панелями, оставляя все остальное место в окне для окон дочерних форм. При этом обычно окно родительской формы в исходном состоянии разворачивают на весь экран. Из родительской формы можно управлять дочерними формами.

Для создания приложения MDI необходимо спроектировать родительскую и дочернюю формы.



Для того чтобы создать MDI приложение необходимо у формы, которую планируется сделать «главной» установить свойство *IsMdiContainer = true*. Тогда она сможет размещать внутри себя дочерние формы. При вызове дочерних форм, чтобы они размещались внутри «главной», необходимо задать «главную» форму в свойстве *MdiParent*.

Для создания дочерней формы необходимо добавить в проект приложения новую форму, которая будет играть роль шаблона для создания дочерних MDI-окон.

В обработчик события на создание нового документа (пункт меню) необходимо добавить код:

```
private void menuItem2_Click(object sender, System.EventArgs e)
{
  Form2 mdiChild = new Form2();
  mdiChild.MdiParent = this;
  mdiChild.Show();
}
```

Если в MDI приложении открыто большое количество дочерних форм, работать с ними может оказаться неудобно. Чтобы хотя бы частично решить данную проблему предусмотрена возможность упорядочения расположения дочерних форм внутри главной. Существуют три типа такого упорядочения:

- Каскадом;
- По горизонтали;
- По вертикали.

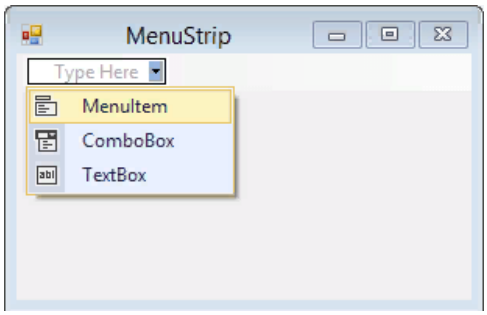
Упорядочение задаётся при помощи метода *LayoutMdi* «главной формы». Этот метод принимает единственный параметр типа *MdiLayout,* который собственно и задаёт тип упорядочения.

Меню

Для создания главного меню приложения в Windows Forms применяется элемент *MenuStrip.* Свойства компонента *MenuStrip:*

- *Dock:* прикрепляет меню к одной из сторон формы
- *LayoutStyle:* задает ориентацию панели меню на форме.
- *ShowItemToolTips:* указывает, будут ли отображаться всплывающие подсказки для отдельных элементов меню
- *Stretch:* позволяет растянуть панель по всей длине контейнера
- *TextDirection:* задает направление текста в пунктах меню.

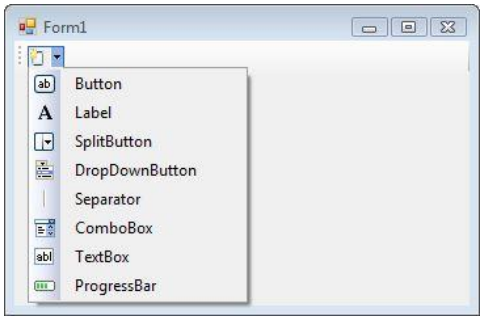
MenuStrip выступает своего рода контейнером для отдельных пунктов меню, которые представлены объектом *ToolStripMenuItem*. Добавить новые элементы в меню можно в режиме дизайнера:



Для добавления доступно три вида элементов*: MenuItem* (объект ToolStripMenuItem), *ComboBox* и *TextBox*. Выпадающие списки и текстовые поля, как правило, применяются в основном на панели инструментов. Меню же обычно содержит набор объектов *ToolStripMenuItem.*

Один элемент *ToolStripMenuItem* может содержать набор других объектов *ToolStripMenuItem*. И таким образом, образуется иерархическое меню или структура в виде дерева.

Для создания инструментального меню, элементы которого чаще всего дублируют функции, которые уже есть в главном меню, и предоставляя пользователю возможность быстро вызвать команду, используют панели *ToolStrip*:



Можно создать следующие типы компонентов

- *Button;*
- *Label;*
- *SplitButton* - кнопка, которая имеет дополнительную кнопку для вызова всплывающего меню;
- *DropDownButton* - кнопка, которая вызывает всплывающее меню;
- *Separator* - разделитель, который позволяет логически разделять группы кнопок;
- *ComboBox* - выпадающий список;
- *TextBox* - текстовое поле ввода;
- *ProgressBar* - индикатор процесса.

Работа с окнами диалога

Операционная система Windows включает много встроенных диалоговых окон, доступ к которым можно получать через API-интерфейс Windows:

- *OpenFileDialog* - выбор имени файла при открытии документа;
- *SaveFileDialog* - задание имени при сохранении документа*;*
- *ColorDialog* - выбор цвета;
- *FontDialog* - для настройки шрифтов*;*
- *FolderBrowserDialog* - работа с каталогами*;*
- *PageSetupDialog* - настройка печати*;*
- *PrintDialog* - организация печати*;*
- *PrintPreviewDialog* - предварительный просмотр документа.

OpenFileDialog и *SaveFileDialog* имеют ряд общих свойств, среди которых можно выделить следующие:

- *DefaultExt:* устанавливает расширение файла, которое добавляется по умолчанию, если пользователь ввел имя файла без расширения;
- *AddExtension*: при значении *true* добавляет к имени файла расширение при его отсутствии (расширение берется из свойства *DefaultExt* или *Filter*);
- *CheckFileExists*: если имеет значение true, то проверяет существование файла с указанным именем;
- *CheckPathExists*: если имеет значение *true*, то проверяет существование пути к файлу с указанным именем;
- *FileName*: возвращает полное имя файла, выбранного в диалоговом окне;
- *Filter:* задает фильтр файлов;
- *InitialDirectory*: устанавливает каталог, который отображается при первом вызове окна;
- *Title*: заголовок диалогового окна.

Обработчик события вызова окна диалога:

```
...
if(Диалог.ShowDialog()== DialogResult.OK)
  {
    // Выполнение каких-либо действий.
    // Обычно — это применение выбранных в окне диалога параметров
  }
...
```

В операторе *if* вызывается метод *ShowDialog()* - показать (открыть) диалог. В открытом окне диалога пользователь выбирает, задаёт тем или иным способом необходимые параметры. Затем он нажимает на кнопку для подтверждения выбора (для разных диалогов она может иметь разное название, например, «Открыть» для диалога по выбору имени открываемого файла) или кнопку «Отмена» для отмены действий. Если пользователь подтвердил свой выбор, то метод *ShowDialog()* возвращает результат *DialogResult.OK*. При отмене действий метод вернёт *DialogResult.Cancel*.

Передача данных через буфер Clipboard

Операция копирования выделенного фрагмента текста из активного MDI-окна в Clipboard (обработчик событий к строке Копирование меню Редактирование или кнопке на панели инструментов):

```
private void menuItem8_Click(object sender, System.EventArgs e)
{
  Form activeChild = this.ActiveMdiChild;
  if(activeChild != null)
  {
    RichTextBox editBox = (RichTextBox)activeChild.ActiveControl;
    if(editBox != null)
    {
      Clipboard.SetDataObject(editBox.SelectedText);
    }
  }
}
```

Операция вставки данных из универсального буфера обмена Clipboard в активное MDI-окно (обработчик события для строки Вставка меню Редактирование):

```
private void menuItem9_Click(object sender, System.EventArgs e)
{
  Form activeChild = this.ActiveMdiChild;
  if(activeChild != null)
  {
    RichTextBox editBox = (RichTextBox)activeChild.ActiveControl;
    if(editBox != null)
    {
      IDataObject data = Clipboard.GetDataObject();
      if(data.GetDataPresent(DataFormats.Text))
      {
        editBox.SelectedText =
          data.GetData(DataFormats.Text).ToString();
      }
    }
  }
}
```

Задание

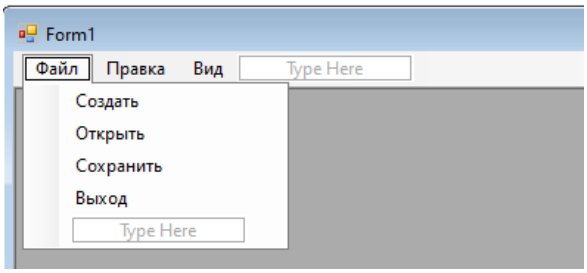
Создать текстовый редактор, применяя технологию разработки многооконного приложения. Основные функции:

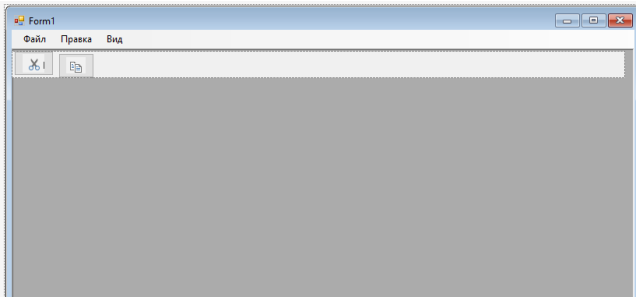
- работа с текстом (ввод, редактирование, форматирование, сохранение);
- работа с файлами (чтение и запись в файл);
- работа с буфером обмена.

Текстовый редактор должен содержать главное меню и инструментальную панель.

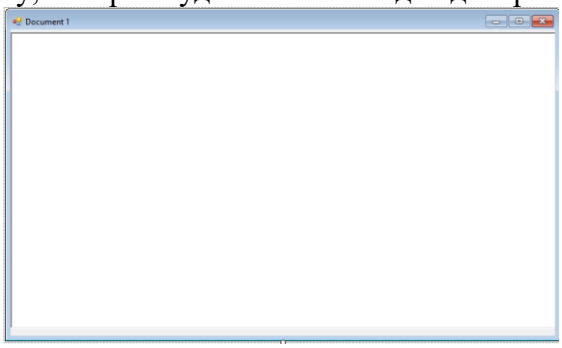
Технология работы

1. Разработку текстового редактора необходимо начать с разработки главной формы:
   - определить размеры и положение формы;
   - задать заголовок формы;
   - определить цветовую гамму;
   - разместить меню и определить пункты.
   - создать меню инструментов.

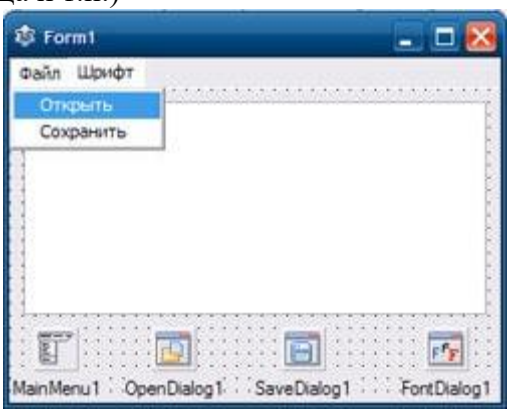




2. Создать новую форму, которая будет шаблоном для дочерних документов.

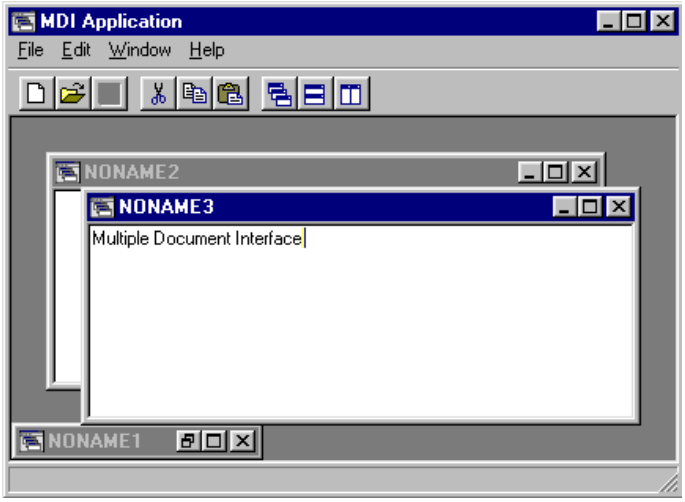


3. Добавить необходимые окна диалогов (открытие, сохранение, шрифты, форматирование абзаца и т.п.)



4. Выполнить настройку свойств элементов управления.
5. Написать соответствующие обработчики событий.
6. Выполнить тестирование приложения.

Пример:



Контрольные вопросы:

1. Объясните термин «визуальное программирование».
2. Какой класс является базовым для всех классов, с помощью которых строится пользовательский интерфейс?
3. Какая форма является главной в проекте?
4. С помощью какого метода можно сделать форму видимой/невидимой?
5. С помощью какого свойства определяется, что форма будет родительской?
6. Опишите технологию создания дочерней формы.