

Министерство науки и высшего образования Российской Федерации
Севастопольский государственный университет
Кафедра ИС

Отчет
по лабораторной работе №4
«Исследование создания mdi-приложений. Сериализация объектов. Стандартные
диалоги»
по дисциплине
«ПЛАТФОРМА .NET»

Выполнил студент группы ИС/б-17-2-о

Горбенко К. Н.

Проверил

Забаштанский А.К.

Севастополь

2019

1 ЦЕЛЬ РАБОТЫ

- Изучить особенности разработки MDI-приложений в Visual Studio .Net;
- изучить способы сохранения данных в файл и загрузки из файла;
- освоить механизм сериализации и десериализации объектов.

2 ЗАДАНИЕ НА РАБОТУ

Для варианта № 3 задана следующая схема данных: **ПЕЧАТНОЕ ИЗДАНИЕ: название, ФИО автора, стоимость.**

1. Создать текстовый редактор NotepadC#, добавив недостающие пункты меню и функции.
2. На основании лабораторной работы 3 создать MDI-приложение. Информация в окне должна отображаться в виде таблицы. Иметь возможность делать выборку данных по различным критериям.
3. Добавить пункты меню для сохранения объектов в файл и загрузки. При сохранении использовать стандартные диалоговые окна и механизм сериализации. В класс добавить поле «Дата создания объекта». Поле не сериализовать, а при десериализации заново устанавливать по системной дате.

3 ХОД РАБОТЫ

3.1 Текстовый редактор Notepad

Создадим текстовый редактор. Основу текстового редактора составляет элемент RichTextBox. RichTextBox не имеет свойства Text, поэтому его нельзя привязать к соответствующему свойству во вью-модели. Решением является использование сторонней библиотеки, расширяющей RichTextBox: XceedWpfToolkit.

Создадим главную страницу приложения:

```

1 <Page x:Class="Notepad.Views.NotepadControl"
2     ...
3     xmlns:viewModels="clr-namespace:Notepad.ViewModels"
4     xmlns:xceedWpfToolkit="clr-namespace:Xceed.Wpf.Toolkit;assembly=Xceed.Wpf
      .Toolkit"
5     d:DataContext="{d:DesignInstance viewModels:NotepadControlViewModel}">
6
7 <DockPanel LastChildFill="True">
```

```

8      <Menu DockPanel.Dock="Top">
9          <MenuItem Header="File">
10             <MenuItem Header="Open" Click="Open" />
11             <MenuItem Header="Save" Click="Save"/>
12             <MenuItem Header="Save As" Click="SaveAs" />
13          </MenuItem>
14      </Menu>
15      <xceedWpfToolkit:RichTextBox Text="{Binding FileContent, Delay=2000,
16          UpdateSourceTrigger=PropertyChanged}"
17          ScrollViewer.HorizontalScrollBarVisibility
18              ="Auto">
19          <FlowDocument PageWidth="3000" />
20          <xceedWpfToolkit:RichTextBox.TextFormatter>
21              <xceedWpfToolkit:PlainTextFormatter />
22          </xceedWpfToolkit:RichTextBox.TextFormatter>
23      </xceedWpfToolkit:RichTextBox>
24  </DockPanel>
25 </Page>

```

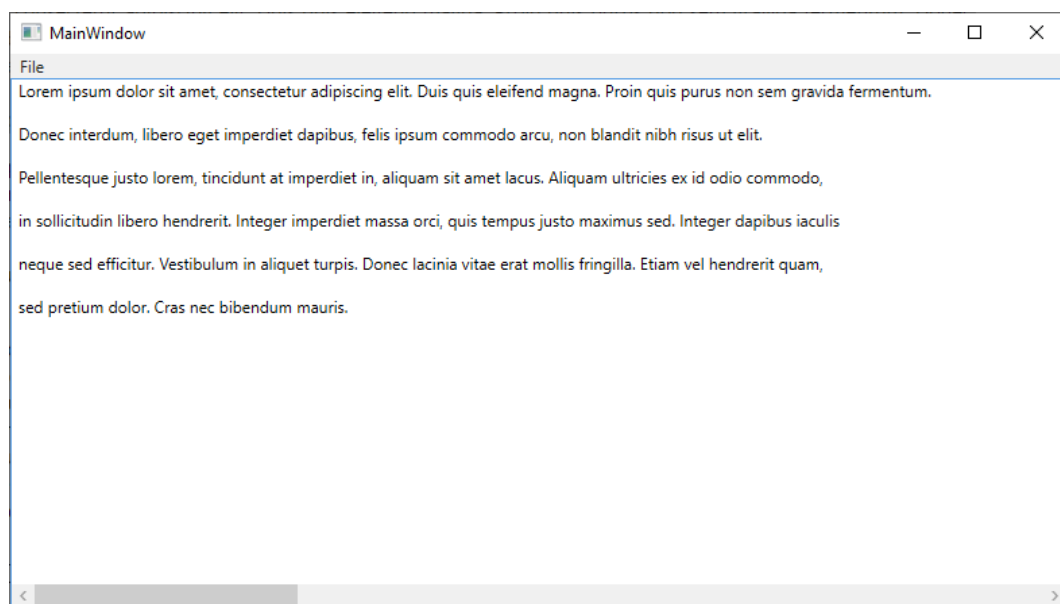


Рисунок 1 – Текстовый редактор

В заголовке страницы импортировано пространство имен включенной в проект библиотеки. Для компоновки страницы использовался элемент `DockPanel` с включенным свойством `LastChildFill`, которое заставляет последний элемент занять все возможное место. Это использовано для того, чтобы растянуть редактор на все окно.

Ниже приведена логика вызова диалогов выбора файла для открытия и сохранения. Кроме того, там происходит вызов команд вью-модели.

```

1 public partial class NotepadControl
2 {
3     public NotepadControl()
4     {
5         InitializeComponent();
6     }
7
8     private void Open(object sender, RoutedEventArgs e)
9     {
10         var dialog = new OpenFileDialog
11         {
12             Filter = "All files (*.*) | *.*"
13         };
14
15         if (dialog.ShowDialog() != true) return;
16
17         if (DataContext is NotepadControlViewModel viewModel)
18         {
19             viewModel.OpenCommand.Execute(dialog.FileName);
20         }
21     }
22
23     private void Save(object sender, RoutedEventArgs e)
24     {
25         if (DataContext is NotepadControlViewModel viewModel)
26         {
27             if (string.IsNullOrEmpty(viewModel.FilePath))
28             {
29                 SaveAs(sender, e);
30             }
31             else
32             {
33                 viewModel.SaveCommand.Execute(null);
34             }
35         }
36     }
37
38     private void SaveAs(object sender, RoutedEventArgs e)
39     {
40         var dialog = new SaveFileDialog
41         {
42             Filter = "All files (*.*) | *.*"
43         };
44
45         if (dialog.ShowDialog() != true) return;
46
47         if (DataContext is NotepadControlViewModel viewModel)
48         {

```

```

49         viewModel.SaveAsCommand.Execute(dialog.FileName);
50     }
51 }
52 }

```

Вью-модель представлена на следующем листинге.

```

1 public class NotepadControlViewModel : INotifyPropertyChanged
2 {
3     private ICommand openCommand;
4     private ICommand saveAsCommand;
5     private ICommand saveCommand;
6     private string fileContent;
7     private string filePath;
8
9     public string FileContent
10    {
11        get => fileContent;
12        set
13        {
14            fileContent = value;
15            OnPropertyChanged(nameof(FileContent));
16        }
17    }
18
19    public string FilePath
20    {
21        get => filePath;
22        set
23        {
24            filePath = value;
25            OnPropertyChanged(nameof(FilePath));
26        }
27    }
28
29    public ICommand OpenCommand =>
30        openCommand ??= new RelayCommand(obj =>
31        {
32            try
33            {
34                if (!(obj is string path)) return;
35
36                FileContent = File.ReadAllText(path);
37                filePath = path;
38            }
39            catch
40            {
41                filePath = string.Empty;
42                throw;

```

```

43         }
44     });
45
46     public ICommand SaveCommand => saveCommand ??= new RelayCommand(obj =>
        SaveAsCommand.Execute(filePath));
47     public ICommand SaveAsCommand => saveAsCommand ??= new RelayCommand(obj =>
48     {
49         if (!(obj is string path)) return;
50
51         File.WriteAllText(path, FileContent);
52         filePath = path;
53     });
54
55     public event PropertyChangedEventHandler PropertyChanged;
56
57     [NotifyPropertyChangedInvocator]
58     protected virtual void OnPropertyChanged([CallerMemberName] string
        propertyName = null)
59     {
60         PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
61     }
62 }

```

В данной вью-модели представлены свойства `OpenCommand`, `SaveAsCommand`, `SaveCommand`, `FileContent`, `FilePath`. `FileContent` и `FilePath` представляют содержание файла и путь к нему соответственно. Содержание обновляется автоматически при изменении данных в редакторе, путь обновляется только при открытии или сохранении.

Команды `OpenCommand`, `SaveAsCommand`, `SaveCommand` необходимы для реализации логики открытия файла и сохранения в файл.

3.2 Mdi приложение

Создадим страницу, представляющую форму для изменения списка объектов и их свойств в виде таблицы. В качестве таблицы используем `DataGrid`. Для таблицы создадим столбцы и свяжем их с соответствующими свойствами печатного издания. Добавим поле даты. Для удаления печатного издания из таблицы создадим столбец с соответствующей кнопкой. Разметка страницы представлена далее:

```

1 <Page Name="PrintedEditionControlPage"
2     xmlns:viewModels="clr-namespace:PrintedEditionMdi.ViewModels"

```

```

3      d:DataContext="{d:DesignInstance viewModels:PrintedEditionControlViewModel
      }"
4      Title="PrintedEditionControl">
5
6      <StackPanel>
7          <Menu>
8              <MenuItem Header="File">
9                  <MenuItem Header="Open" Click="OpenButtonClick"></MenuItem>
10                 <MenuItem Header="Save As" Click="SaveAsButtonClick"></MenuItem>
11             </MenuItem>
12         </Menu>
13         <TextBox Text="{Binding Filter, UpdateSourceTrigger=PropertyChanged}"
14             FlowDirection="RightToLeft" Margin="5"></TextBox>
15         <DataGrid HorizontalAlignment="Left"
16             ItemsSource="{Binding PrintedEditions}"
17             SelectedItem="{Binding SelectedItem}"
18             AutoGenerateColumns="False"
19             IsTextSearchEnabled="True">
20             <DataGrid.Columns>
21                 <DataGridTextColumn Binding="{Binding Name}" Width="0.2*"
22                     Header="Name" />
23                 <DataGridTextColumn Binding="{Binding Author}" Width="0.2*"
24                     Header="Author" />
25                 <DataGridTextColumn Binding="{Binding Price}" Width="0.2*"
26                     Header="Price" />
27                 <DataGridTextColumn Binding="{Binding CreatedAt, StringFormat='
28                     dd/MM/yyyy'}" IsReadOnly="True"
29                     Width="0.2*" Header="Created at" />
30                 <DataGridTemplateColumn Width="0.1*">
31                     <DataGridTemplateColumn.CellTemplate>
32                         <DataTemplate>
33                             <Button Command="{Binding Path=DataContext.
34                                 RemoveCommand,
35                                     ElementName=
36                                         PrintedEditionControlPage
37                                         }">Delete</Button>
38                         </DataTemplate>
39                     </DataGridTemplateColumn.CellTemplate>
40                 </DataGridTemplateColumn>
41             </DataGrid.Columns>
42         </DataGrid>
43     </StackPanel>
44 </Page>

```

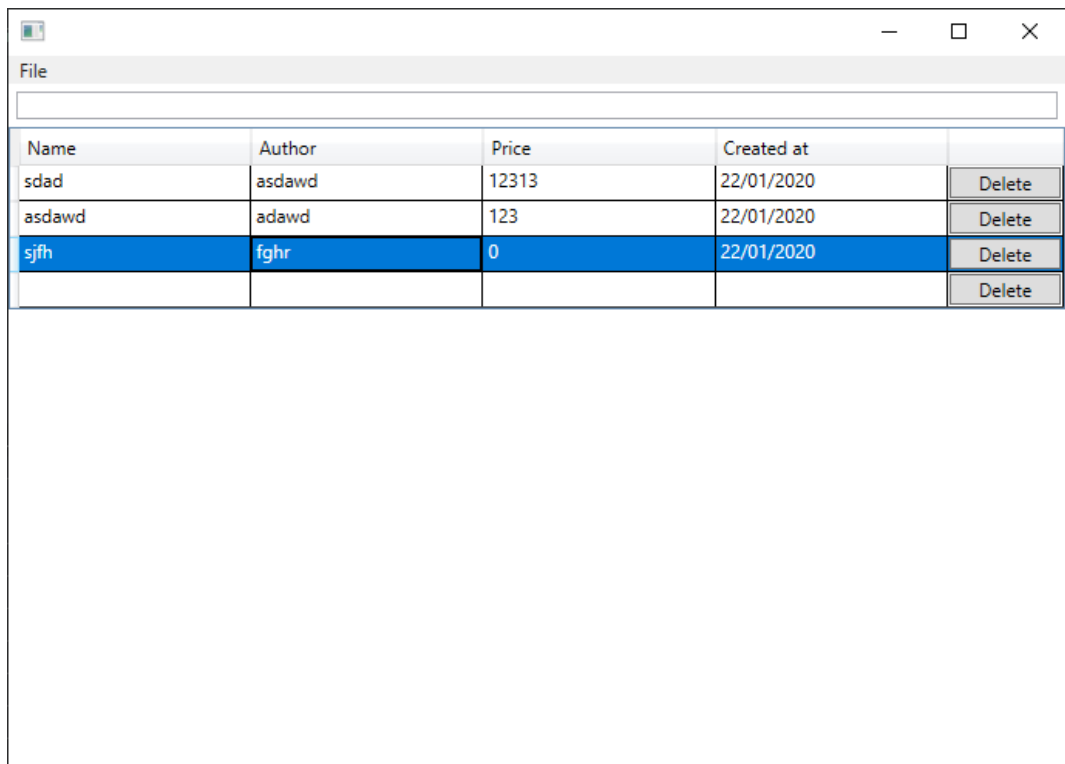


Рисунок 2 – Приложение редактирования печатных изданий

Рассмотрим вызов команд `OpenButtonClick` и `SaveAsButtonClick`.

```

1 public partial class PrintedEditionControl
2 {
3     public PrintedEditionControl()
4     {
5         InitializeComponent();
6     }
7
8     private void SaveAsButtonClick(object sender, RoutedEventArgs e)
9     {
10         var dialog = new SaveFileDialog
11         {
12             FileName = "PrintedEditions",
13             DefaultExt = ".json",
14             Filter = "Printed edition json (.json)|*.json"
15         };
16
17         if (dialog.ShowDialog() != true) return;
18
19         if (DataContext is PrintedEditionControlViewModel viewModel &&
20             viewModel.SaveCommand.CanExecute(new object()))
21         {
22             viewModel.SaveCommand.Execute(dialog.FileName);
23         }
24     }

```



```

25
26 private void OpenButtonClick(object sender, RoutedEventArgs e)
27 {
28     var dialog = new OpenFileDialog
29     {
30         Filter = "Printed edition json (.json)|*.json"
31     };
32
33     if (dialog.ShowDialog() != true) return;
34
35     if (DataContext is PrintedEditionControlViewModel viewModel &&
36         viewModel.OpenCommand.CanExecute(new object()))
37     {
38         viewModel.OpenCommand.Execute(dialog.FileName);
39     }
40 }
41 }

```

В данных методах происходит вызов диалогов выбора файла для открытия и для сохранения в файл. Затем, в случае успешного выбора, вызывается соответствующая команда вью-модели. Рассмотрим вью-модель.

```

1 public class PrintedEditionControlViewModel : INotifyPropertyChanged
2 {
3     private readonly ICollectionView printedEditionsView;
4     private ObservableCollection<PrintedEdition> printedEditions;
5     private PrintedEdition selectedItem;
6     private string filter;
7     private ICommand removeCommand;
8     private ICommand saveCommand;
9     private ICommand openCommand;
10
11     public event PropertyChangedEventHandler PropertyChanged;
12
13     [NotifyPropertyChangedInvocator]
14     private void OnPropertyChanged([CallerMemberName] string propertyName =
15         null) =>
16         PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
17
18     public PrintedEditionControlViewModel()
19     {
20         printedEditions = new ObservableCollection<PrintedEdition>();
21         printedEditionsView = CollectionViewSource.GetDefaultView(
22             printedEditions);
23         printedEditionsView.Filter = x =>
24         {
25             if (string.IsNullOrEmpty(filter)) return true;

```

```

24         if (!(x is PrintedEdition printedEdition)) return false;
25
26         return printedEdition.ToString().Contains(filter);
27     };
28 }
29
30 public ObservableCollection<PrintedEdition> PrintedEditions
31 {
32     get => printedEditions;
33     set
34     {
35         printedEditions = value;
36         printedEditionsView.Refresh();
37         OnPropertyChanged(nameof(PrintedEditions));
38     }
39 }
40
41 public PrintedEdition SelectedItem
42 {
43     get => selectedItem;
44     set
45     {
46         selectedItem = value;
47         OnPropertyChanged(nameof(SelectedItem));
48     }
49 }
50
51 public string Filter
52 {
53     get => filter;
54     set
55     {
56         if (filter == value) return;
57
58         filter = value;
59         printedEditionsView.Refresh();
60         OnPropertyChanged(nameof(Filter));
61     }
62 }
63
64 public ICommand RemoveCommand =>
65     removeCommand ??= new RelayCommand(obj => printedEditions.Remove(
66         selectedItem));
67
68 public ICommand SaveCommand =>
69     saveCommand ??= new RelayCommand(obj =>
70         PrintedEditionSerializeHelper.Serialize(printedEditions, obj as
71             string));

```

```

70
71     public ICommand OpenCommand =>
72         openCommand ??= new RelayCommand(obj =>
73             {
74                 var currentDateAndTime = DateTime.Now;
75
76                 PrintedEditionSerializeHelper.Deserialize(obj as string)
77                     .Select(x => { x.CreatedAt =
78                         currentDateAndTime; return x;
79                     })
80                     .ForEach(x => PrintedEditions.Add(x));
81             });
82 }

```

Для реализации фильтрации используется тип `ICollectionView`, в котором есть функция фильтрации. Фильтр задается привязкой к полю в представлении. В командах открытия и сохранения из файла используется механизм сериализации в `Json`. Его реализация находится в классе `PrintedEditionSerializeHelper`. Его структура:

```

1 public static class PrintedEditionSerializeHelper
2 {
3     public static IEnumerable<PrintedEdition> Deserialize([NotNull] string path
4         )
5     {
6         if (path == null) throw new ArgumentNullException(nameof(path));
7
8         using var streamReader = File.OpenText(path);
9         var json = streamReader.ReadToEnd();
10
11         return JsonConvert.DeserializeObject<PrintedEdition[]>(json);
12     }
13
14     public static void Serialize([NotNull] IEnumerable<PrintedEdition>
15         printedEditions, [NotNull] string path)
16     {
17         if (printedEditions == null) throw new ArgumentNullException(nameof(
18             printedEditions));
19         if (path == null) throw new ArgumentNullException(nameof(path));
20
21         var json = JsonConvert.SerializeObject(printedEditions, Formatting.
22             Indented);
23
24         using var streamWriter = new StreamWriter(path);
25         streamWriter.Write(json);
26     }
27 }

```

Данный класс выполняет сериализацию и десериализацию коллекции печатных изданий в Json, а затем записывает результат в файл.

ВЫВОД

В ходе лабораторной работы было создано Mdi-приложение с табличным редактором, возможностью записи данных в файл и открытием файла. Для создания таблицы использовался элемент `DataGrid`, имеющий возможность привязки данных в таблице к свойству класса C#. Для сериализации использовалась библиотека `Newtonsoft.Json`.

Кроме того, был создан текстовый редактор на основе `RichTextBox`.