Министерство Образования Республики Беларусь
Белорусский Государственный Университет
Информатики и Радиоэлектроники
Кафедра Информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ

Дисциплина: Программирование

Тема: Разработка блокнота для цитат

Принял:

Козуб Виктор Николаевич

Выполнил: студент гр.653501

Хоронеко Максим Павлович

Минск 2017

Содержание

Постановка цели

Конечной целью этого проекта является создание блокнота для цитат. Простого, при этом с достаточной для большинства пользователей функциональностью. Возможностью быстрого добавления цитат, авто-удалением по истечении срока, установленного юзером, мгновенным ручным удалением. Так же должна существовать возможность прикрепления изображений к цитатам. Дизайн приложения должен быть неброским и вписываться в любое оформление Windows, так как возможно, что блокнот будет активен на протяжении всего сеанса работы с ОС.

В ходе работы над этим проектом на практике закрепить основы разработки приложений в WPF (Windows Presentation Foundation). Выявить основные проблемы, характерные для работы над подобными приложениями, тем самым определить знания и навыки, необходимые для их решения. Это должно упростить более глубокое изучение данной системы в будущем.

Сравнение с Microsoft OneNote 2016

1. Функциональность

Notes значительно уступает продукту Microsoft в функциональности. В OneNote реализован форматированный ввод текста, рисование, вставка звуковых заметок, изображений, таблиц, цитат в разном виде. У его младшего аналога в свою очередь реализована вставка изображений в заметки и некоторые манипуляции с цитатами.

2. Интерфейс

Обе программы обладают достаточно дружелюбным интерфейсом. Однако стоит отметить, что в связи с повышенной функциональностью аналог от компании Microsoft вызывает чувство некоторой нагромождённости, чего нельзя сказать о Notes. При этом концепция свободного рабочего поля OneNote заставляет пользователя разворачивать приложение на достаточно большую область экрана, что в некоторых случаях может быть неудобно, при этом у его аналога такой необходимости никогда не возникает.

3. Производительность.

В активном состоянии Microsoft OneNote 2016 потребляет в среднем 40 мб оперативной памяти. Как и ожидалось, у проекта Notes этот показатель значительно ниже: около 17 мб.

В скрытом состоянии Microsoft OneNote 2016 потребляет около 0.1 мб оперативной памяти, когда как Notes по-прежнему использует 17 мб. В этом отношении второй проигрывает.

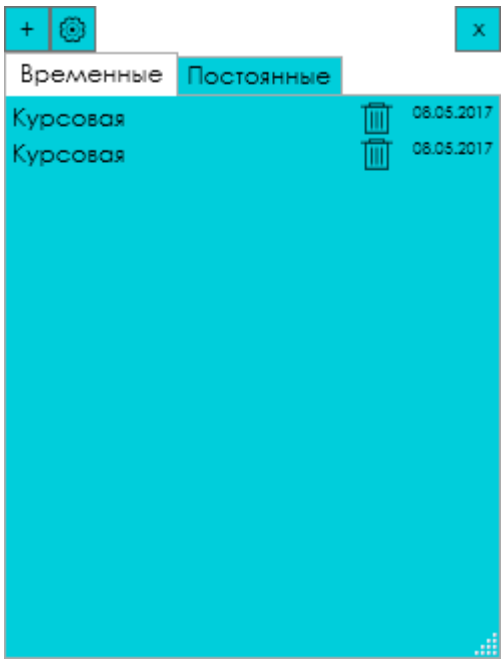
Обе программы показали приемлемый результат, они не будут значительным образом влиять на производительность системы. Подобные отличия не так важны в современных реалиях.

4. История

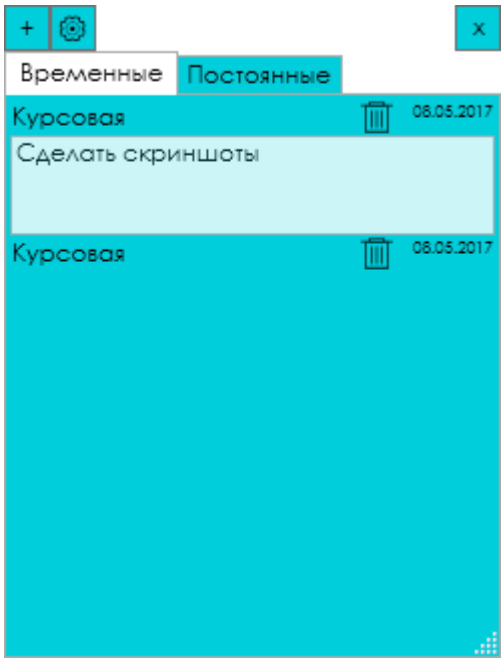
Первое публичное заявление о Microsoft OneNote (тогда Microsoft Office OneNote) датируется 17 ноября 2002 года. Продукт появился в связи с развитием платформы Microsoft Tablet PC. Впервые был выпущен в 2003 году в составе Microsoft Office System 2003. 17 марта 2014 года программа стала бесплатной, её можно скачать отдельно с сайта Microsoft.

Что же касается Notes, на его разработку было потрачено около трёх недель. В случае необходимости работа над ним может быть продолжена, так как приложение довольно легко поддается расширению за счет небольшого количества составляющих.

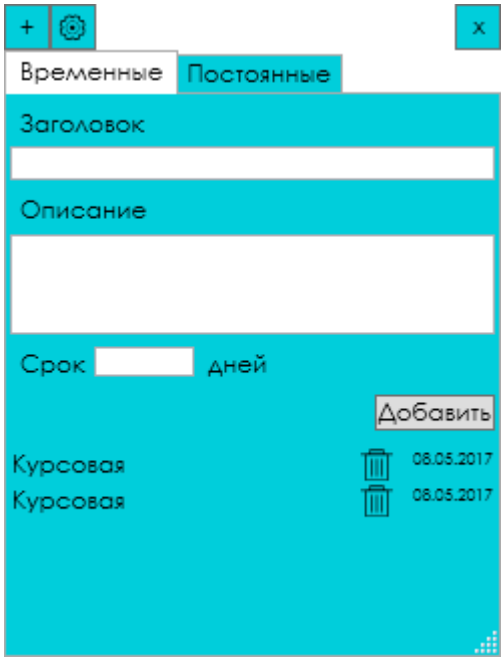
Скриншоты и описание



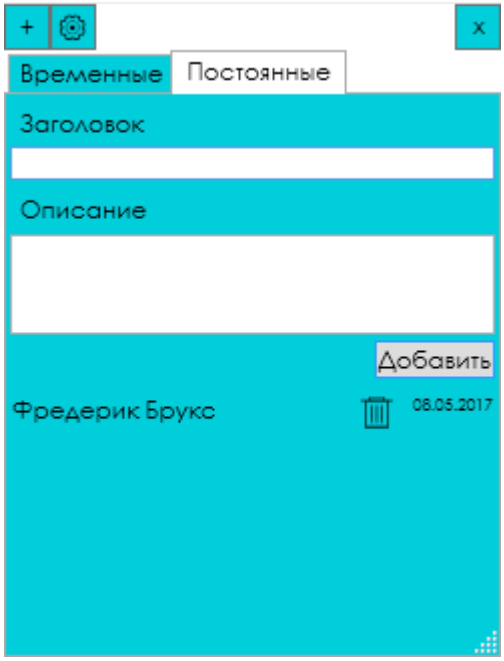
Интерфейс программы довольно простой и интуитивно понятный. Кнопка «+» служит для добавления новой заметки. Шестеренка для отображения настроек. Крестик для закрытия программы. Две вкладки позволяют выбирать между заметками с временным хранением и постоянным. Значок корзины служит для удаления ненужных заметок.



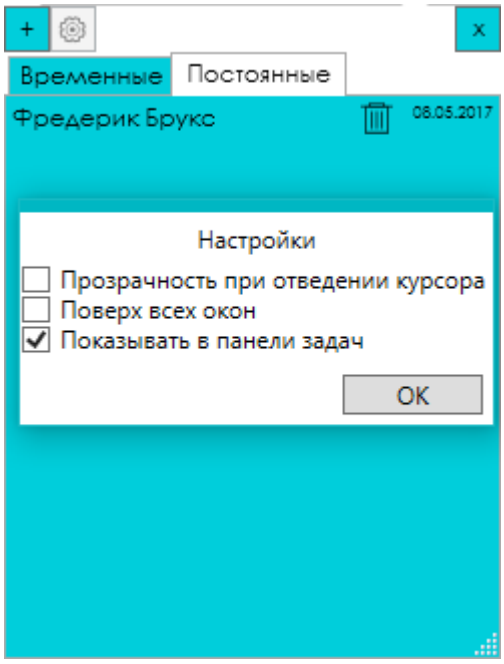
Заметки раскрываются и скрываются по нажатии на заголовок.



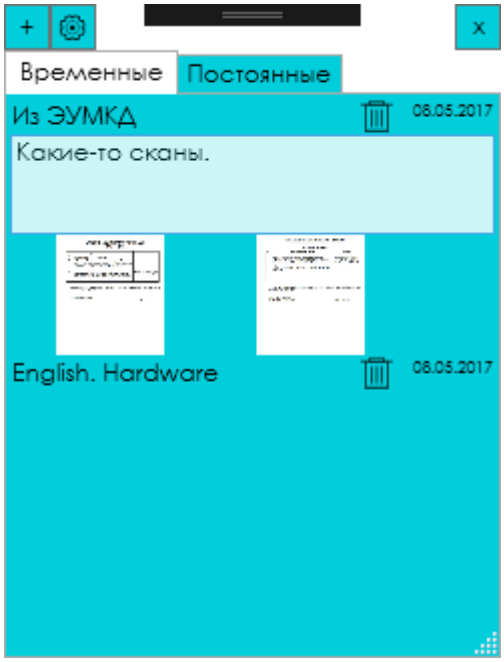
При нажатии на «+» с открытой вкладкой «Временные» появляется блок, показанный на скриншоте выше. Для его вызова может также служить комбинация клавиш Ctrl+N. Срок можно не указывать, тогда заметка автоматически отправится в «Постоянные». Можно также не указывать заголовок или описание.



Блок добавления для постоянных цитат.



При нажатии на шестерню открывается окно настроек, где можно выбрать, включать ли прозрачность, держать ли блокнот с заметками поверх всех окон, и показывать ли его значок в панели задач.



К заметкам можно прикреплять изображения путем перетаскивания их на нужный заголовок. Программа поддерживает добавление нескольких картинок одним переносом. Изображения можно раскрыть в полный размер, кликнув по ним, и таким же образом закрыть.

Техническая информация

Средства разработки

Для разработки приложения использовалась система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем WPF (Windows Presentation Foundation), графическая подсистема в составе .NET Framework, использующая язык XAML.

WPF предустановлена в Windows Vista, 7, 8, 10. С её помощью можно создавать широкий спектр как автономных, так и запускаемых в браузере приложений.

Разработка велась в среде Visual Studio 2017, так как данная среда является новейшим полноценным средством работы с платформой .NET и поставляется бесплатно в редакции Community.

Структура программы

MainWindow.xaml – тут содержится код xaml разметки главного окна приложения.

MainWindow.xaml.cs – файл с кодом C#, в проекте в нем описаны действия при открытии и закрытии программы.

Add.cs – код, обрабатывающий событие нажатия кнопки «+».

Buttons.cs – код, обрабатывающий нажатие кнопок «x» и настроек.

Drop.cs – код для поддержки прикрепления изображений посредствам перетаскивания их на заметку.

File.cs – файл с логикой сохранения заметок и настроек в файлы путем сериализации.

Focus.cs – обработка движения мыши из главного окна и в него.

Note.cs – файл с описанием класса Note.

OpenPicture.cs – открытие картинки в полный размер и ее закрытие.

Settings.cs – код окна настроек.

Описание работы программного кода

При запуске программы сначала загружаются настройки:

```
MyFile.LoadSettings();
```

LoadSettings() является методом статического класса MyFile, находящегося в файле File.cs.

```
public static SettingsInfo LoadSettings()
        {
            BinaryFormatter formatter = new BinaryFormatter();
            SettingsInfo s = new SettingsInfo();
            using (FileStream fs = new FileStream("Settings.dat", FileMode.OpenOrCreate))
            {
                try
                {
                    s = (SettingsInfo)formatter.Deserialize(fs);
                }
                catch
                {
                    return null;
                }
            }
            return s;
        }
```

Для сохранения и загрузки настроек используется механизм сериализации объектов. В .Net реализованы XAML, Soap, JSON сериализация, а также бинарная, которую мы и использовали. Можно было использовать любую из них, кроме XAML, так как последняя не сериализует статические поля, которые использовались в классе SettingsInfo, представленном в файле Settings.cs.

```
[Serializable]
    class SettingsInfo : ISerializable
    {
        public static bool Transparency { get; set; }
        public static bool OnTop { get; set; }
        public static bool InTaskbar { get; set; }


        public SettingsInfo()
        {

        }

        public void GetObjectData(SerializationInfo info, StreamingContext context)
        {
            info.AddValue("static.Transparency", SettingsInfo.Transparency,
typeof(bool));
```

```
            info.AddValue("static.OnTop", SettingsInfo.OnTop, typeof(bool));
            info.AddValue("static.InTaskbar", SettingsInfo.InTaskbar, typeof(bool));
        }

        public SettingsInfo(SerializationInfo info, StreamingContext context)
        {
            SettingsInfo.Transparency = info.GetBoolean("static.Transparency");
            SettingsInfo.OnTop = info.GetBoolean("static.OnTop");
            SettingsInfo.InTaskbar = info.GetBoolean("static.InTaskbar");
        }
    }
```

Этот класс отмечен атрибутом [Serializable], который указывает на возможность сериализации класса. Обычно этого хватает, но в данном случае, так как в классе содержатся статические поля, необходимо реализовать интерфейс ISerializable, который позволяет объектам управлять их собственной сериализацией и десериализацией. Метод GetObjectData позволяет внести в файл именованные значения, которые при десериализации будут считаны в конструкторе с соответствующими параметрами.

После того, как настройки были загружены, происходит инициализация списка заметок.

```
Note.SetList(MyFile.Open());
```

Статический метод Note.Setlist(List<Note>) определен в классе Note и просто присваивает статическому полю список, полученный при десериализации в методе MyFile.Open(), анадогичной той, что используется для получения настроек.

Далее мы получаем цвет, который используется для закрашивания границ окна в Windows и для повышения читаемости черного шрифта, делаем этот цвет светлее, если есть такая необходимость

```
Color windowColor = SystemParameters.WindowGlassColor;
            if (windowColor.R < 200 && windowColor.G < 200 && windowColor.B < 200)
            {
                windowColor.R = (byte)(windowColor.R * 1.2);
                windowColor.G = (byte)(windowColor.G * 1.2);
                windowColor.B = (byte)(windowColor.B * 1.2);
            }
```

```
            SolidColorBrush windowBrush = new SolidColorBrush(windowColor);
```

После осветления, создается кисть, использующая этот цвет

```
Style backStyle = new Style();
            backStyle.Setters.Add(new Setter(BackgroundProperty, windowBrush));
            backStyle.Setters.Add(new Setter(FontFamilyProperty, new FontFamily("Century
Gothic MS")));
            this.Resources.Add("Color", backStyle);
            Actual.Style = Permanent.Style = Type.Style = Adder.Style = Closer.Style =
Settings.Style = backStyle;
```

Ко всем элементам управления и панелям применяется стиль с цветом оформления Windows и шрифтом Century Gothic.

После этого из уже загруженного ранее списка заметок проверяется, какие из них больше не актуальны, в связи с истечением срока, установленного пользователем, остальные выводятся на панель, их визуализация осуществляется при помощи функции Note.NoteVisualize(Note)

```
List<int> toRemove = new List<int>();

            int i = 0;
            if (notes != null)
                foreach (Note note in notes)
                {
                    if (note.type == Notes.Type.Actual)
                    {
                        if(DateTime.Now >= note.CollapseDate)
                        {
                            toRemove.Add(i);
                            continue;
                        }
                        ActualNotes.Children.Add(Note.NoteVisualize(note));
                    }
                    else
                    {
                        PermNotes.Children.Add(Note.NoteVisualize(note));
                    }

                    i++;
                }
            else
                Note.SetList(new List<Note>());

            foreach(int n in toRemove)
                notes.RemoveAt(n);
```

Логика прикрепления картинок к заметкам находится в файле Drop.cs. В нем определен метод Note_Drop(object, DragEventArgs), который является ответом на событие DropEvent (перетаскивание на элемент) определенной заметки. Обработка события начинается с выставления эффекта при перетаскивании на копирование, после чего объявляются и инициализируются переменные, содержащие копируемые данные, список с изображениями, так как оно может быть не одно и коллекция строк, в которой будут храниться пути к файлам.

```
e.Effects = DragDropEffects.Copy;
var data = e.Data as DataObject;
var imgList = new List<Image>();
StringCollection files = new StringCollection();
```

Далее из каждого файла при помощи функции LoadImageFromFile(string path) в список imgList добавляется изображение, каждому из которых инкрементируется обработчик события нажатия левой кнопки мыши, который позволяет открыть картинку в полном размере.

```
if (data.ContainsFileDropList())
                    {
                        files = data.GetFileDropList();
                        for (int j = 0; j < files.Count; j++)
                        {
                           imgList.Add(new Image());
                           imgList[imgList.Count - 1].Source
                           LoadImageFromFile(files[j]);
                           imgList[imgList.Count - 1].MouseLeftButtonDown +=
                           AttachedImage_FullSize;
                        }

                    }
```

Вся информация о заметках хранится в коллекции List<Note> noteList. При закрытии программы все заметки считываются с графического интерфейса заново, сохраняются в noteList и сериализуются в файл Notelist.dat.

```
Note.SetList(list);
MyFile.Save(Note.GetList());
```

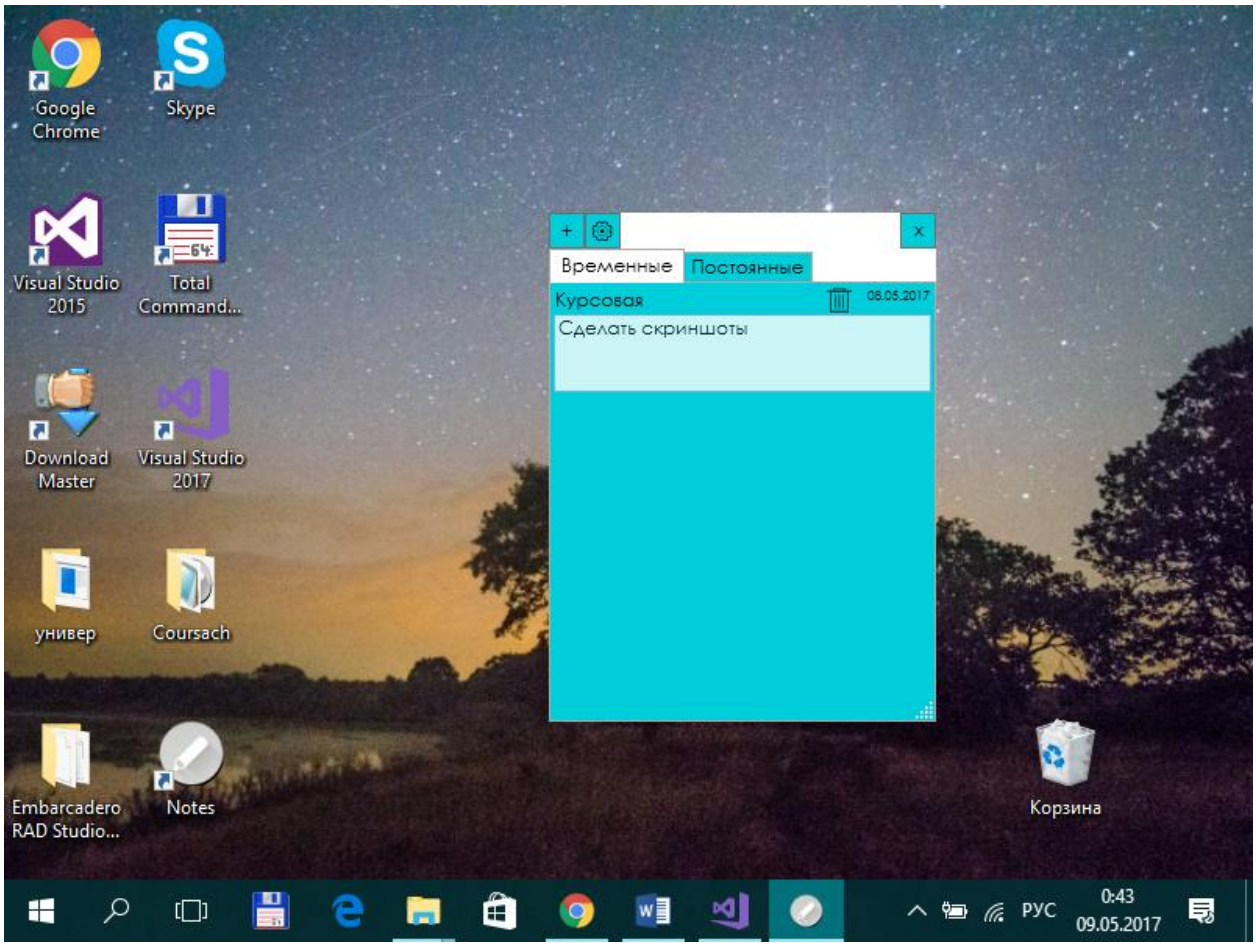
Настройки также сохраняются при помощи механизма сериализации

```
MyFile.SaveSettings(new SettingsInfo());
```

Тестирование

Работа программы была проверена на операционных системах Windows 7, 8, 10. Как показали тесты, приложение ведет себя предсказуемо вне зависимости от версии ОС Windows. Текст остается читаемым и графическое оформление вписывается в любую цветовую схему, предложенную Microsoft.

Также работа программы была проверена на всех возможных разрешениях экрана, что также подтвердило ее стабильность. Благодаря языку разметки XAML (eXtensible Application Markup Language), приложение является аппаратно-независимым, так как в XAML название «пиксель» лишь условное, потому что представляет собой величину, равную 1/96 дюйма, что позволяет без труда создавать приложения, устойчивые к изменению разрешения. Скриншот рабочего стола при разрешении 800x600:



Выводы

В ходе работы над проектом были изучены и закреплены на практике основные принципы работы в WPF. Был выявлен ряд проблем, часть из которых была решена во время работы над приложением. Среди этих проблем: сериализация объектов, содержащих статические члены, некоторые нюансы работы с изображениями в WPF, особенности взаимодействия кода C# с языком разметки XAML, а также событийной модели в WPF.

При этом также были определены проблемы, требующие дальнейшего изучения. Например, сохранение чистоты и, соответственно, обслуживаемости кода. В данном проекте это не совсем получилось, из-за отсутствия какого-либо опыта работы в WPF в прошлом, недостаточного времени, отведенного на проектирование программы и неполное знание средств выбранной системы. Также стала очевидной необходимость придерживаться заранее определенного стиля написания кода для облегчения его читаемости.

Несмотря на некоторые недостатки в программном коде, поставленные цели были достигнуты. Программа имеет ненавязчивый дизайн и определенный изначально функционал, работает стабильно, не зависимо от программного и аппаратного обеспечения. Недостатки, выявленные при написании кода, определяют направление изучения мною не только данной системы, но и принципов проектирования ПО в целом, поэтому проект считаю полезным и успешно завершенным.

Список литературы

1. https://ru.wikipedia.org/wiki/Windows_Presentation_Foundation - Википедия. Свободная энциклопедия.
2. https://metanit.com/sharp/wpf/ - Metanit. Сайт о программировании.
3. https://ru.wikipedia.org/wiki/XAML - Википедия. Свободная энциклопедия.
4. https://msdn.microsoft.com/ru-ru/library/ms744948(v=vs.110).aspx – Microsoft Developer Network.
5. https://habrahabr.ru/post/137405/ - Хабрахабр.
6. https://ru.wikipedia.org/wiki/Microsoft_OneNote - Википедия. Свободная энциклопедия.