Лабораторная работа №3:
Разработка приложения “Таймер”

Цель:

Целью данной работы является закрепление навыков использования компонентов и описания событий на языке высокого уровня C# в среде программирования Microsoft Visual Studio 2022 Community

Задания:

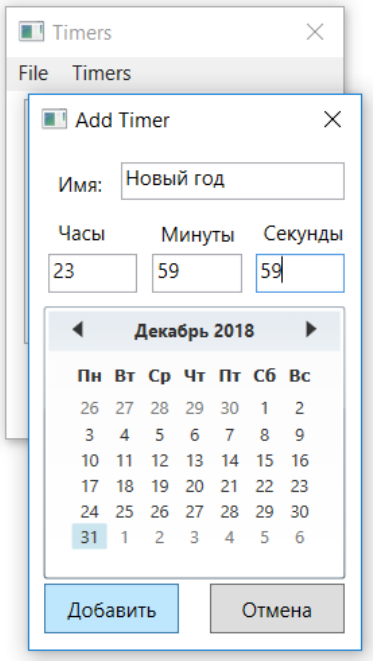
Разработать программное приложение, имеющее графический интерфейс и реализующее следующие функции:

1. Добавление “Таймера”. Под добавлением таймера подразумевается ввод или выбор пользователем даты, до которой требуется отсчитывать время, а также, ввод имени, ассоциированного с этим таймером. Добавленные таймеры должны отображаться на форме приложения в виде списка.
2. Выбор таймера из списка и отображение времени, оставшегося до наступления указанной в нём даты, в формате: дни/часы/минуты/секунды
3. Возможность редактирования и удаления таймеров из списка.
4. Возможность сохранения списка таймеров в текстовый файл.
5. Возможность загрузки списка таймеров из текстового файла.

Дополнительно:

1. Добавление таймера должно выполняться при помощи диалогового окна.
2. Возможность выбрать формат отображения оставшегося времени:
   - дни/часы/минуты/секунды
   - часы/минуты/секунды (сколько всего часов минут и секунд осталось)
   - минуты/секунды (сколько всего минут и секунд осталось)
   - секунды (сколько всего секунд осталось)
3. Возможность свернуть программу в область уведомлений

Пример внешнего вида формы добавления таймера:



Справочная информация:

В этом разделе описаны компоненты, классы и методики, которые могут быть полезны при выполнении лабораторной работы.

Для хранения и операцией над датой и временем, могут быть использованы классы DateTime и TimeSpan.

Класс DateTime (дата и время) содержит в себе следующие свойства и методы:

`Year, Month, Day, Hour, Minute, Second` – свойства, содержащие информацию о записанных в класс: годе, месяце, часе, минуте и секунде соответственно.

`Add` – метод, позволяющий добавить к записанной в классе дате промежуток времени в формате TimeSpan.

`Parse` – метод, преобразующий строку содержащую дату и время в объект типа DateTime.

`Now` – метод, возвращающий текущее время (установленное в операционной системе).

Вывесли

```
DateTime dt = new DateTime(); //создание переменной типа Дата и Время
dt = dt.AddSeconds(356);      //добавление к значению переменной 350 секунд
MessageBox.Show("Время: " + dt.ToString("HH:mm:ss")); //вывод на экран времени в формате “чч:мм:сс”
```

Класс TimeSpan (отрезок времени) содержит в себе следующие свойства и методы:

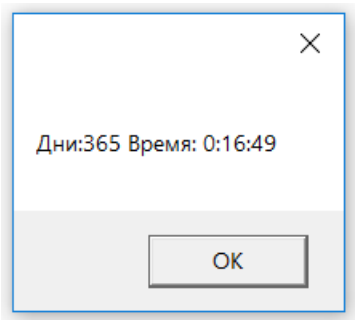
`Days, Hours, Minutes, Seconds` – свойства, содержащие информацию о количестве: месяцев, часов, минут и секунд, содержащихся в отрезке времени.

`TotalDays, TotalHours, TotalMinutes, TotalSeconds` – свойства, содержащие информацию об отрезке времени в: месяцах, часах, минутах и секундах соответственно.

Пример вычисления отрезка времени между двумя датами:

```
//устанавливается дата - 28.02.2020 21:30:10
DateTime d1 = new DateTime(2020, 2, 28, 21, 30, 10);
//устанавливается текущая дата
DateTime d2 = DateTime.Now;
//вычисляется время между двумя датами
TimeSpan ts = d1 - d2;
//вывод на экран времени, между датой d2 и датой d1
MessageBox.Show("Дни:"+ts.Days+" Время: "+ts.Hours+":"+ts.Minutes+":"+ts.Seconds);
```

Результат работы:



Компонент Calendar (календарь) содержит в себе следующие свойства:

`DisplayDateStart, DisplayDateEnd` – задают соответственно начальную и конечную дату диапазона, который будет отображаться в календаре.

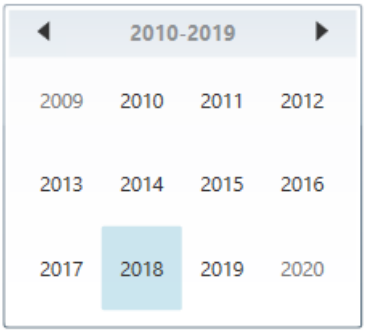
`IsTodayHighlighted` – отмечает, будет ли выделена текущая дата.

`DisplayMode` – определяет формат отображения календаря. Может принимать одно из следующих значений:

- `Month` (по умолчанию) отображает все дни текущего месяца
- `Decade` отображает все года текущего десятилетия
- `Year` отображает все месяцы текущего года

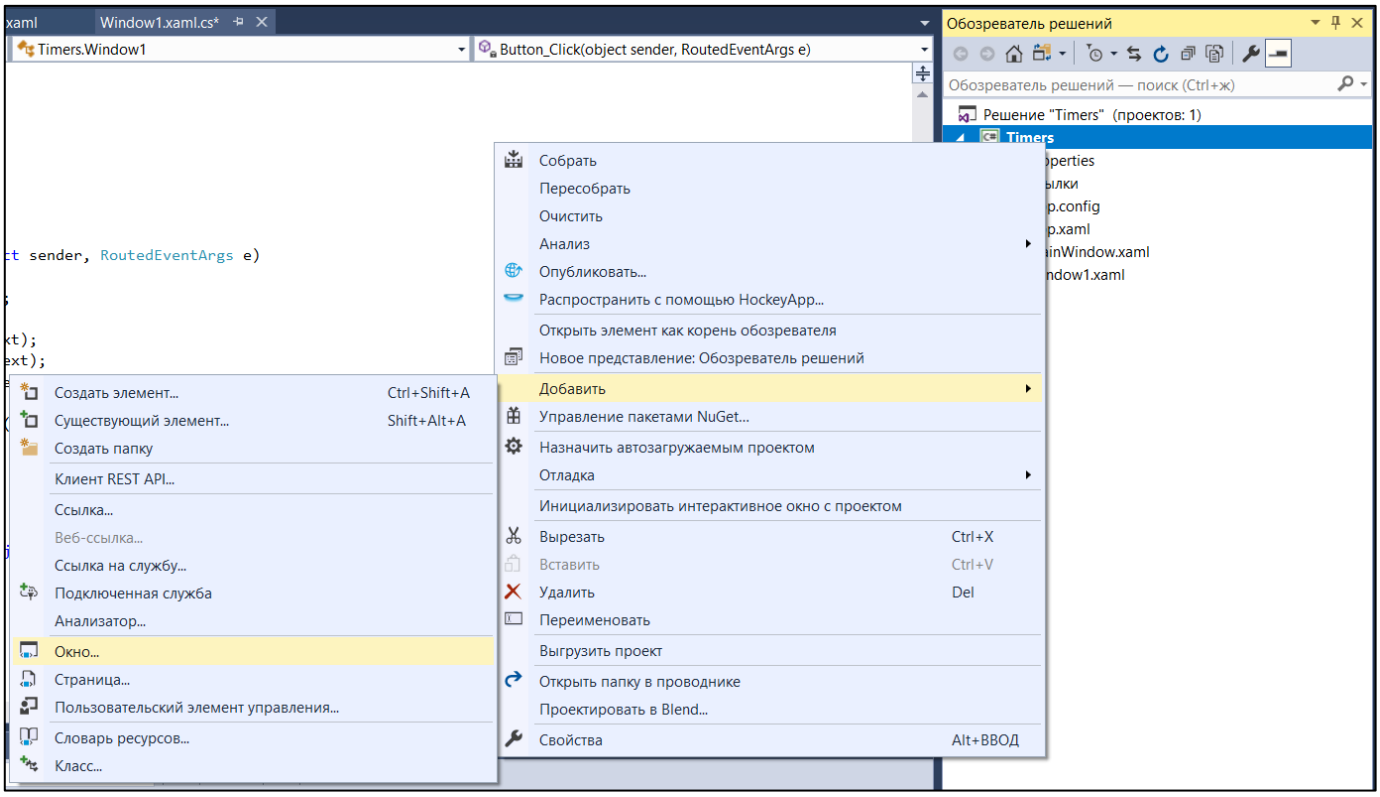
`SelectedDate.Value` – содержит выбранное значение в формате DateTime.

Внешний вид компонента, с установленным значением `DisplayMode = CalendarMode.Decade`:

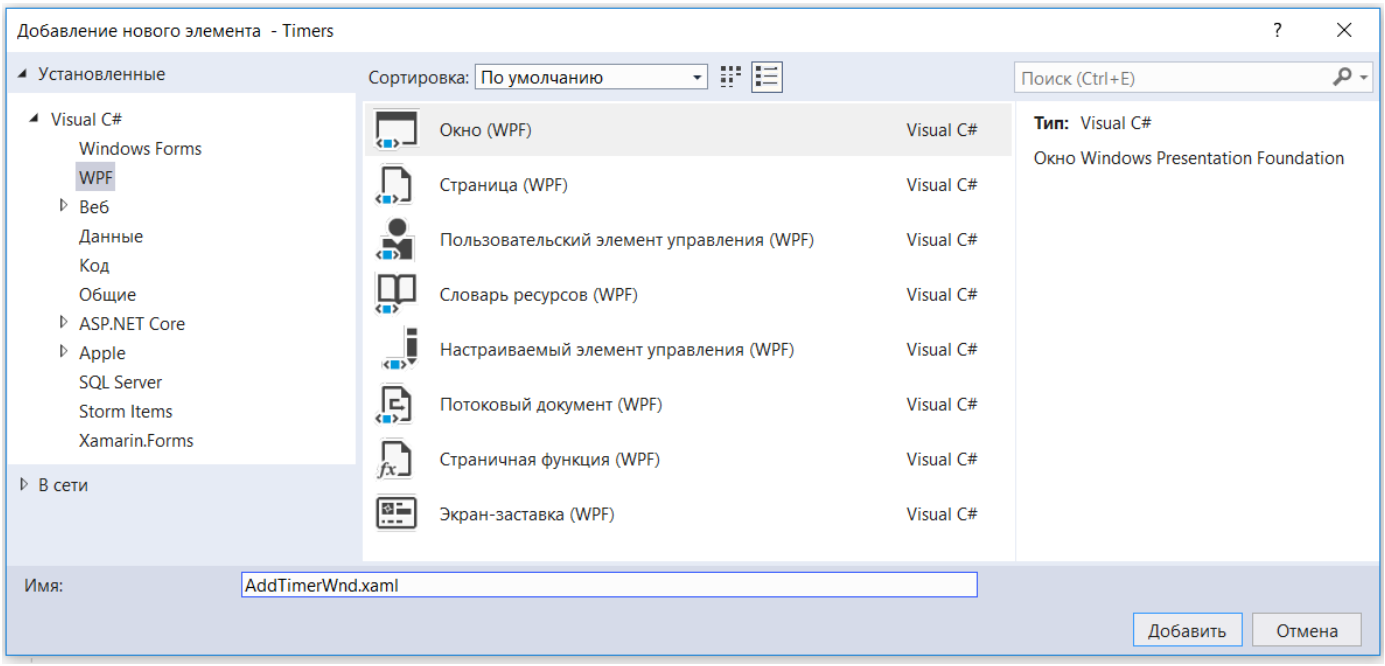


Добавление диалогового окна:

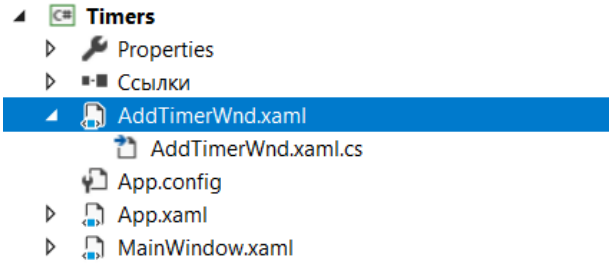
Для того, чтобы добавить дополнительное окно в ваш проект, нажмите правой кнопкой мыши на название проекта в обозревателе решений, выберите Добавить -> Окно:



После чего выберите Окно (WPF) и введите название:



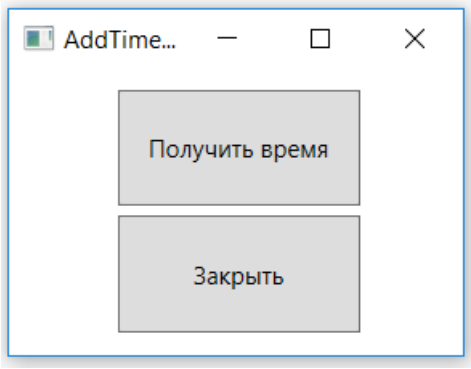
В проект будет автоматически добавлены новые XAML и CS файлы, содержащие описание окна и исходный код с ним связанный:



Пример диалогового окна, получающего текущее время:

```
public partial class AddTimerWnd : Window
{
    //создание глобальной переменной для хранения даты и времени
    public DateTime current;

    public AddTimerWnd()
    {
        InitializeComponent();
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        //при нажатии кнопки “Получить время”, текущее время записывается
        //в глобальную переменную current
        current = DateTime.Now;
        //закрытие окна с отметкой об успешном завершении работы
        this.DialogResult = true;
    }

    private void Button_Click_1(object sender, RoutedEventArgs e)
    {
        //при нажатии кнопки “Закрыть” происходит
        //закрытие окна с отметкой об не успешном завершении работы
        this.DialogResult = false;
    }
}
```

Внешний вид диалогового окна:



Для того, чтобы вызвать диалоговое окно из основного, и получить от него данные, можно использовать код следующего вида:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    //создание нового окна (название класса – то, что было указано при добавлении окна)
    AddTimerWnd add_timer = new AddTimerWnd();

    //вызов окна + проверка, отработало ли окно корректно
    if (add_timer.ShowDialog() == true)
    {
        //поскольку переменная current, окна add_timer была объявлена как public
        //можно обратиться к ней напрямую и получить необходимые данные
        DateTime st = add_timer.current;
    }
    else //если окно отработало с результатом false
    {
        //либо вывести сообщение, что данные не были получены
        //либо ничего не делать
    }
}
```

Класс Dictionary<> (словарь):

Словарь является подвидом массива данных. Основным отличием является способ доступа к хранимым данным, если у массива это индексы, идущие по порядку, то у словаря это объекты произвольного типа, например, строки.

Пример использования:

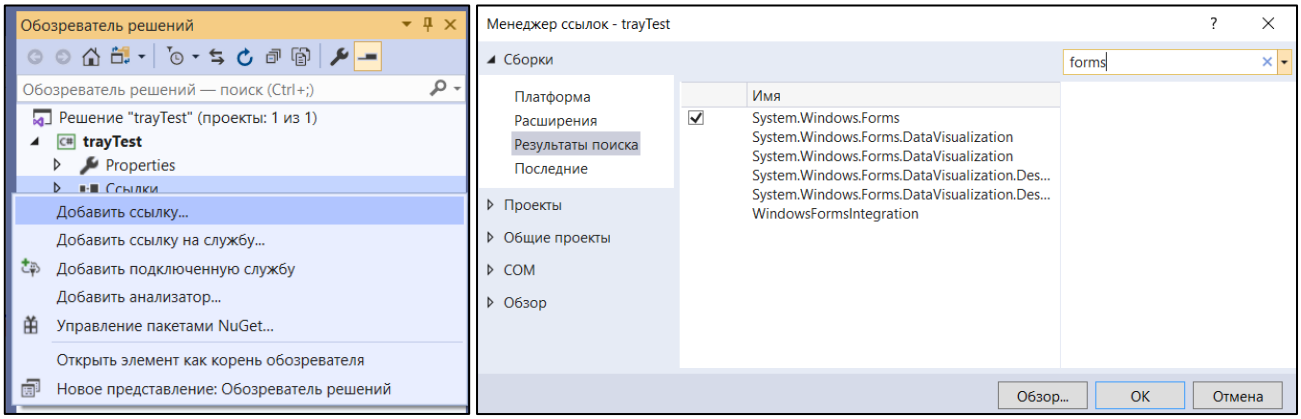
```
//создание словаря для хранения дат в формате DateTime
//ключом словаря является строка
Dictionary<string, DateTime> list = new Dictionary<string, DateTime>();

//добавление в словарь даты 25.05.2018 8:00:00 под именем “начало сессии”
list.Add(“начало сессии”, new DateTime(2018, 5, 25, 8, 0, 0));

//получение из словаря даты, хранящейся под именем “начало сессии”
DateTime d = list[“начало сессии”];


//удаление из словаря даты, хранящейся под именем “начало сессии”
list.Remove("начало сессии");
```

В данной лабораторной работе, можно хранить таймеры в словаре, используя в качестве ключа их название, а в качестве значения – дату и время.

Сворачивание окна в область уведомлений:

Для программ отсчёта времени, характерна возможность работы в резидентном режиме (работает постоянно на фоне). Для того, чтобы скрыть программу с рабочего стола и панели “Пуск”, можно отображать её работу в области уведомлений:

Сделать это можно, добавив к проекту ссылки на пространства имён Forms и Drawing:



А затем, добавить к проекту код следующего вида:

```
System.Windows.Forms.NotifyIcon ni = new System.Windows.Forms.NotifyIcon();

public MainWindow()
{
    InitializeComponent();

    //загрузка картинки, которая будет отображаться в области уведомлений
    ni.Icon = new System.Drawing.Icon(@"Timer.ico");
    ni.Visible = true;
    //обработчик события "двойной клик" по иконке в области уведомлений
    ni.DoubleClick +=
        delegate (object sender, EventArgs args)
        {
            //показать окно
            this.Show();
            this.WindowState = WindowState.Normal;
        };
}
//перезапись метода обработки события сворачивания окна
protected override void OnStateChanged(EventArgs e)
{
    //вместо того что бы сворачивать окно, обработчик события будет его скрывать
    if (WindowState == System.Windows.WindowState.Minimized)
        this.Hide();

    base.OnStateChanged(e);
}
```

В результате, при попытке свернуть окно, оно будет отправляться в панель уведомлений. При двойном щелчке по иконке в панели уведомлений, окно будет разворачиваться к исходному виду.

Для того, чтобы загрузить иконку из ресурсов приложения, можно использовать функцию:

```
ni.Icon = System.Drawing.Icon.ExtractAssociatedIcon("Timer.ico");
```

Для того, чтобы вывести текст в область уведомлений с задержкой в 5000мс, можно использовать:

```
ni.ShowBalloonTip(5000, "Title", "Text", System.Windows.Forms.ToolTipIcon.Info);
```

Список литературы:

Структура DateTime: https://msdn.microsoft.com/ru-ru/library/system.datetime(v=vs.110).aspx

Форматы перевода DateTime в строку: https://www.c-sharpcorner.com/blogs/date-and-time-format-in-c-sharp-programming1

Структура TimeSpan: https://msdn.microsoft.com/ru-ru/library/system.timespan(v=vs.110).aspx

Форматы перевода TimeSpan в строку: https://docs.microsoft.com/ru-ru/dotnet/standard/base-types/custom-timespan-format-strings

Класс Calendar:

https://msdn.microsoft.com/ru-ru/library/system.windows.controls.calendar(v=vs.110).aspx

О диалоговых окнах:

https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/app-development/dialog-boxes-overview

Класс Dictionary: https://msdn.microsoft.com/ru-ru/library/xfhwa508(v=vs.90).aspx

Учебник. Создание первого приложения WPF в Visual Studio 2019:

https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/getting-started/walkthrough-my-first-wpf-desktop-application