МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

ФГБОУ ВО «БАЙКАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра математических методов и цифровых технологий

**Разработка приложений для мобильных устройств**

Отчёт по первой лабораторной работе

|  |  |  |
| --- | --- | --- |
| студента бакалавриата группы | БИ-17-1 | Точилова Е.А. Фамилия И.О. |
| проверил | к.т.н., доцент кафедры математических методов и цифровых технологий  ученое звание, должность | Родионов А.В.  Фамилия И.О |

Иркутск, 2020

**Задание к лабораторной работе:**

**TextBlock**

Элемент TextBlock служит для вывода простой текстовой информации, то есть представляет собой обычную метку. За установку текста метки отвечает свойство Text:

|  |
| --- |
| <TextBlock Text="Universal Windows Platfom" /> |

Для настройки шрифта мы можем использовать стандартные свойства FontSize (высота шрифта), FontFamily (семейство шрифтов), FontStretch (растяжение шрифта), FontStyle (стиль наклона), FontWeight (вес/толщина шрифта):

|  |
| --- |
| <TextBlock Text="Universal Windows Platfom" FontSize="20" FontFamily="Verdana"          FontStretch="Normal" FontStyle="Normal" FontWeight="Bold"/> |

С помощью свойств LineHeight, TextWrapping и TextAllignment мы можем задать дополнительные настройки текста.

Свойство LineHeight позволяет указывать высоту строк.

**TextWrapping**

Свойство TextWrapping позволяет переносить текст. Может принимать три значения:

* NoWrap: текст не переносится. Значение по умолчанию
* Wrap: если ширина текстовой метки не позволяет вместить весь текст, то в ней создается новые строки, на которые переносится текст
* WrapWholeWords: текст переносится по границам слов

Установка в коде:

|  |  |  |
| --- | --- | --- |
|  |  | textBlock.TextWrapping = TextWrapping.WrapWholeWords; |

**TextAllignment**

Свойство TextAllignment выравнивает текст. Может принимать следующие значения:

* Center: выравнивание по центру
* DetectFromContent: направление выравнивания зависит от текста
* Justify: текст не растягивается так, чтобы символы были оптимальным образом распределены по всей ширине блока
* Left: выравнивание по левому краю
* Right: выравнивание по правому краю

Установка в коде:

|  |  |
| --- | --- |
|  | textBlock.TextAlignment = TextAlignment.Justify; |

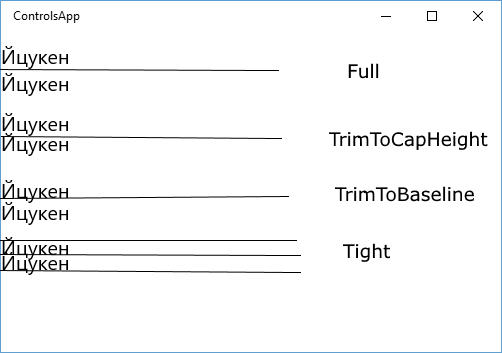
Например:

|  |  |
| --- | --- |
|  | <TextBlock Text="Онегин был по мненью многих ученый малый, но педант. Имел он счастливый талант..."          TextWrapping="Wrap"  LineHeight="30" TextAlignment="Justify" FontSize="20" /> |

**TextLineBounds**

Свойство TextLineBounds устанавливает способ определения высоты для каждой строки текста в элементе. Может принимать следующие значения:

* Full: строка имеет стандартную высоту. Значение по умолчанию
* TrimToCapHeight: высота строки устанавливается по высоте шапки шрифта.
* TrimToBaseline: нижний уровень строки устанавливается по опорной линии текста.
* Tight: высота строки устанавливается по шапке шрифта, а нижний уровень - по опорной линии текста



Установка в коде:

|  |
| --- |
| textBlock.TextLineBounds = TextLineBounds.Full; |

**TextTrimming**

Свойство TextTrimming устанавливает, как будет усекаться текст при выходе за пределы ограничивающего прямоугольника. Может принимать следующие значения:

* None: текст не усекается. Значение по умолчанию
* CharacterEllipsis: текст усекается по границе символа. На месте усечения вставляется многоточие
* WordEllipsis: текст усекается по границе слова. На месте усечения вставляется многоточие
* Clip: текст обрезается на уровне пикселей

Установка в коде:

|  |  |
| --- | --- |
|  | textBlock.TextTrimming = TextTrimming.WordEllipsis; |

**TextReadingOrder и FlowDirection**

Свойство FlowDirection задает направление текста - слева направо или справа налево:

* LeftToRight: направление текста слева направо, значение по умолчанию
* RightToLeft: направление текста справа налево

Установка в коде:

|  |  |
| --- | --- |
|  | textBlock.FlowDirection = FlowDirection.RightToLeft; |

Свойство TextReadingOrder также влияет на порядок чтения:

* Default: по умолчанию не переопределяет направление содержимого. Направление задается через FlowDirection
* DetectFromContent: корректирует порядок, в котором размещаются блоки двунаправленного текста
* UseFlowDirection: то же самое, что и Default

Установка в коде:

|  |  |
| --- | --- |
|  | textBlock.TextReadingOrder = TextReadingOrder.UseFlowDirection; |

**TextBox**

Для ввода текста используется элемент TextBox. Он также, как и TextBlock, имеет свойства TextWrapping, TextAllignment и TextReadingOrder. Для задания шрифта текста также применяются свойства FontFamily, FontSize. Среди других свойств следует выделить MaxLength и PlaceholderText

MaxLength устанавливает предельное количество вводимых символов.

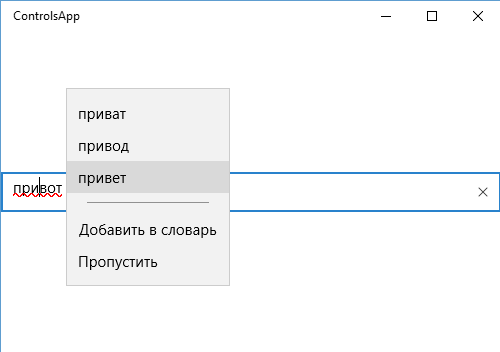
Свойство PlaceholderText задает текст по умолчанию, который отображается в поле до ввода в него символов:

|  |  |
| --- | --- |
|  | <TextBox MaxLength="250" PlaceholderText="Введите текст" Height="40" /> |

Еще одно интересное свойство IsSpellCheckEnabled позволяет автоматически проверять орфографию:

|  |  |
| --- | --- |
|  | <TextBox IsSpellCheckEnabled="True" Height="40" /> |

Если мы введем некорректное значение, то текстовое поле его подчеркнет:



Из событий элемента следует выделить событие TextChanged, которое срабатывает при изменении текста в текстовом поле. Например, подключим обработчик события:

|  |  |
| --- | --- |
|  | <StackPanel>      <TextBox x:Name="textBox1" PlaceholderText="Введите текст" Height="40" TextChanged="TextBox\_TextChanged" />      <TextBlock x:Name="textBlock1" Height="40" />  </StackPanel> |

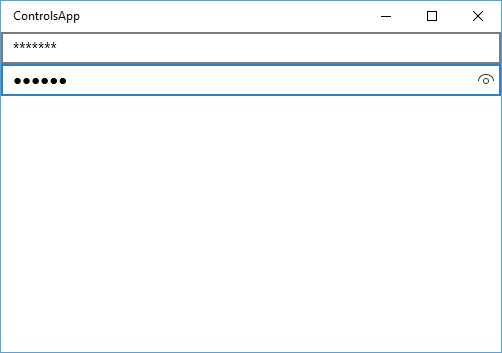
А в файле кода пропишем сам обработчик:

|  |  |
| --- | --- |
|  | private void TextBox\_TextChanged(object sender, TextChangedEventArgs e)  {      string text = textBox1.Text; // получаем введенный текст      textBlock1.Text = text;  } |

**PasswordBox**

PasswordBox применяется для ввода парольной информации. Это тоже самое текстовое поле, только для ввода символов используется маска. Свойство PasswordChar устанавливает символ маски, отображаемый при вводе пароля. Свойство Password задает парольную строку, отображаемую по умолчанию при загрузке окна приложения.

|  |  |
| --- | --- |
|  | <StackPanel>      <PasswordBox PasswordChar="\*" Password="12345" />      <PasswordBox Password="12345" />  </StackPanel> |



**Button**

Элемент Button представляет кнопку:

|  |  |
| --- | --- |
|  | <Button x:Name="button1" Width="100" Height="40" Background="LightGray" Content="Hello UWP" /> |

В основном кнопки служат для инициирования пользователем какого-либо действия по нажатию. Чтобы обработать нажатие кнопки, надо установить для ее события Click кнопки обработчик. Например, код xaml:

|  |  |
| --- | --- |
|  | <Button x:Name="button1" Width="100" Height="40" Background="LightGray" Content="Hello UWP" Click="Button\_Click" /> |

Тогда в файле кода c# для данной кнопки можно было бы прописать обработчик:

|  |  |
| --- | --- |
|  | private void Button\_Click(object sender, RoutedEventArgs e)  {      button1.Content = "Clicked!";  } |

Создание аналогичной кнопки в коде C#:

|  |  |
| --- | --- |
|  | Button button2 = new Button();  button2.Width = 100;  button2.Height = 40;  button2.Content = "Hello UWP";  button2.Background = new SolidColorBrush(Windows.UI.Colors.LightGray);  button2.Click+=Button\_Click; |

**ToggleButton**

ToggleButton представляет простой переключатель, который может пребывать в трех состояниях - отмеченном, неотмеченном и неопределенном. Состояние можно установить или получить с помощью свойства IsChecked. Оно принимает три значения: True (отмеченное состояние), False (неотмеченное состояние) и x:Null (или null в коде c#) - неопределенное состояние.

Чтобы кнопка могла переключаться во все три состояния, у ней надо установить свойство IsThreeState="True", иначе для нее будут доступны только два первых состояния:

|  |  |
| --- | --- |
|  | <StackPanel VerticalAlignment="Top">      <ToggleButton x:Name="toggleButton1" IsThreeState="True" Content="Переключить" Click="ToggleButton\_Click" />      <TextBlock x:Name="textBlock1" />  </StackPanel> |

И в файле кода c# зададим вывод в текстовый блок состояния toggleButton1:

|  |  |
| --- | --- |
|  | private void ToggleButton\_Click(object sender, RoutedEventArgs e)  {      textBlock1.Text = toggleButton1.IsChecked.ToString();  } |



Как правило, ToggleButton сама по себе редко используется, чаще применяются производные классы в виде CheckBox и RadioButton.

**HyperlinkButton**

HyperlinkButton предназначен для вывода ссылок, во всем остальном это обычная кнопка. Для установки ссылки в элементе определено свойство NavigateUri:

|  |  |
| --- | --- |
|  | <HyperlinkButton NavigateUri="metanit.com" Content="Перейти на сайт" Foreground="Red" /> |



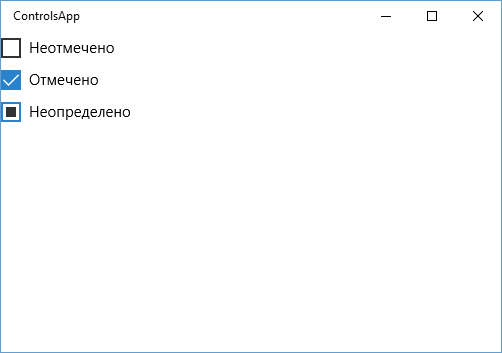
При нажатии на ссылку открывается браузер Microsoft Edge, который переходит по ссылке.

**CheckBox**

Флажок или CheckBox применяется для выбора из двух вариантов по принципу "да-нет", а также когда нам надо выбрать несколько опций из группы опций.

Формально CheckBox может принимать три состояния: Checked, Unchecked и Intermediate. Для установки состояния применяется свойство IsChecked:

|  |  |
| --- | --- |
|  | <StackPanel>      <CheckBox x:Name="checkBox1" IsChecked="False" IsThreeState="True" Content="Неотмечено" />      <CheckBox x:Name="checkBox2" IsChecked="True" Content="Отмечено" />      <CheckBox x:Name="checkBox3" IsChecked="{x:Null}" IsThreeState="True" Content="Неопределено"/>  </StackPanel> |



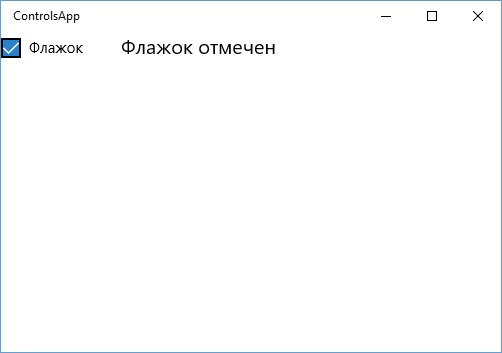
Значение свойства IsChecked="{x:Null}" устанавливает флажок в неопределенное состояние. Остальные два состояния задаются с помощью значений true и false. Кроме того, чтобы можно было переключаться между всеми тремя состояниями, надо установить свойство IsThreeState="True"

С помощью событий Checked (генерируется при установке флажка в отмеченное состояние), Unchecked (генерируется при снятии отметки с флажка) и Indeterminate (флажок переведен в неопределенное состояние) мы можем отследить изменение состояния флажка. Например, определим элемент checkBox:

|  |  |
| --- | --- |
|  | <CheckBox x:Name="checkBox" IsThreeState="True" Content="Неопределено"          Unchecked="checkBox\_Unchecked"          Checked="checkBox\_Checked"          Indeterminate="checkBox\_Indeterminate" /> |

И в файле кода c# определим все обработчики:

|  |  |
| --- | --- |
|  | private void checkBox\_Checked(object sender, RoutedEventArgs e)  {      textBlock1.Text = checkBox.Content.ToString() + " отмечен";  }    private void checkBox\_Unchecked(object sender, RoutedEventArgs e)  {      textBlock1.Text = checkBox.Content.ToString() + " не отмечен";  }    private void checkBox\_Indeterminate(object sender, RoutedEventArgs e)  {      textBlock1.Text = checkBox.Content.ToString() + " в неопределенном состоянии";  } |



Также для нас представляет интерес еще два события - Tapped и Click, которые возникают при нажатии на флажок. Мы можем обработать одно из них:

|  |  |
| --- | --- |
|  | <StackPanel Orientation="Horizontal" VerticalAlignment="Top">      <CheckBox x:Name="checkBox" IsChecked="False" Height="20" Content="Флажок" Tapped="checkBox\_Tapped" />      <TextBlock x:Name="textBlock1" />  </StackPanel> |
|  | private void checkBox\_Tapped(object sender, TappedRoutedEventArgs e)  {      textBlock1.Text = checkBox.IsChecked.ToString();  } |

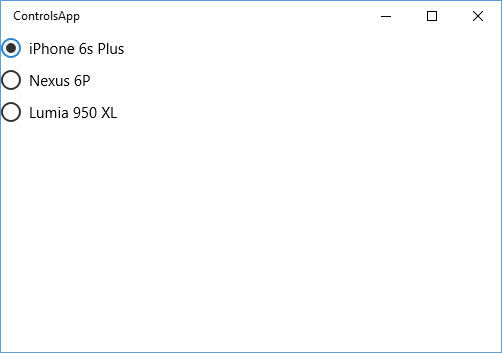
Создание флажка в коде c#:

|  |  |
| --- | --- |
| 1  2  3 | CheckBox chBox = new CheckBox();  chBox.IsChecked = true;  chBox.Content = "Новый флажок"; |

**RadioButton**

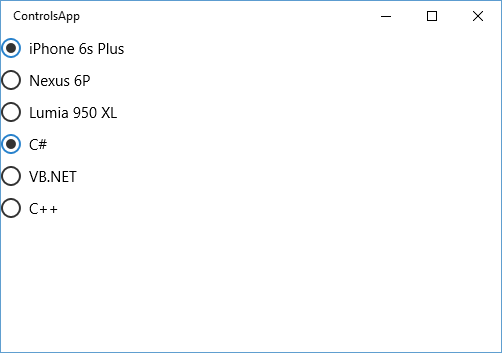
RadioButton представляет переключатель или радиокнопку. Переключатель используется для выбора из двух и более взаимоисключающих вариантов. Элемент RadioButton, как и CheckBox, может быть в отмеченном, неотмеченном и неопределенном состоянии. Отличительной чертой этих элементов является свойство GroupName, которое задает группу радиокнопок. И в один момент времени мы можем отметить в одной группе только одну радиокнопку. Например,

|  |  |
| --- | --- |
|  | <StackPanel>      <RadioButton GroupName="phones" Content="iPhone 6s Plus" IsChecked="True" />      <RadioButton GroupName="phones" Content="Nexus 6P" />      <RadioButton GroupName="phones" Content="Lumia 950 XL" />  </StackPanel> |



Если бы радиокнопки принадлежали бы к двум разным группам, то соответственно мы могли бы выбрать две радиокнопки:

|  |  |
| --- | --- |
|  | <StackPanel>      <RadioButton GroupName="phones" Content="iPhone 6s Plus" IsChecked="True" />      <RadioButton GroupName="phones" Content="Nexus 6P" />      <RadioButton GroupName="phones" Content="Lumia 950 XL" />      <RadioButton GroupName="Languages" Content="C#" IsChecked="True" />      <RadioButton GroupName="Languages" Content="VB.NET" />      <RadioButton GroupName="Languages" Content="C++" />  </StackPanel> |

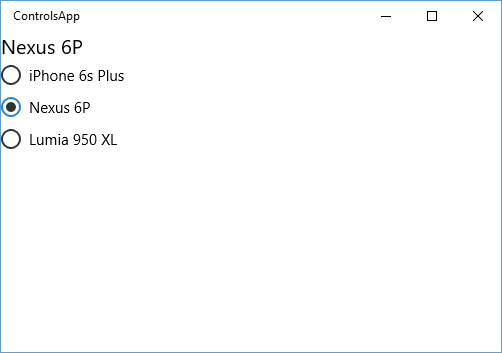


Чтобы проследить за выбором того или иного элемента, мы также можем определить у элементов событие Checked и его обрабатывать в коде:

|  |  |
| --- | --- |
|  | <StackPanel>      <TextBlock x:Name="textBlock1" />      <RadioButton GroupName="phones" Content="iPhone 6s Plus" IsChecked="True" Checked="RadioButton\_Checked" />      <RadioButton GroupName="phones" Content="Nexus 6P" Checked="RadioButton\_Checked" />      <RadioButton GroupName="phones" Content="Lumia 950 XL" Checked="RadioButton\_Checked" />  </StackPanel> |

А в файле кода C# зададим обработчик:

|  |  |
| --- | --- |
|  | private void RadioButton\_Checked(object sender, RoutedEventArgs e)  {      RadioButton pressed = (RadioButton)sender;      textBlock1.Text = pressed.Content.ToString();  } |



**ToggleSwitch**

ToggleSwitch служит для переключения между двумя состояниями, которые вступают в силу сразу после изменения состояния переключателя. В этом плане элемент похож на CheckBox. Для определения текущего состояния используется свойство IsOn:

|  |  |
| --- | --- |
|  | <ToggleSwitch IsOn="True" Header="On/Off" VerticalAlignment="Top" /> |



Свойство Header позволяет задать заголовок. При этом кроме заголовка элемент хранит текстовые метки для включенного и выключенного состояний. По умолчанию это "on" и "off". При выборе текстовые метки отображаются рядом с элементом. Но с помощью свойств OnContent и OffContent мы можем изменить эти метки или вложить более сложное содержимое, чем обычный текст:

|  |  |
| --- | --- |
|  | <ToggleSwitch IsOn="True" OnContent="Включено" OffContent="Выключено" VerticalAlignment="Top" /> |



Несмотря на то, что можно задать длинные метки в качестве текста, рекомендуется использовать короткие метки для обозначения состояния, не длиннее 4 символов, например, вкл. и выкл.

Для отслеживания изменения состояния элементы мы можем задействовать событие Toggled:

|  |  |
| --- | --- |
|  | <StackPanel>      <TextBlock x:Name="textBlock1" />      <ToggleSwitch x:Name="toggleSwitch1" Toggled="toggleSwitch1\_Toggled" IsOn="True"          OnContent="Включено" OffContent="Выключено" />  </StackPanel> |

А в коде c# пропишем обработчик:

|  |  |
| --- | --- |
|  | private void toggleSwitch1\_Toggled(object sender, RoutedEventArgs e)  {      textBlock1.Text = toggleSwitch1.IsOn.ToString();  } |



Программное создание ToggleSwitch:

|  |  |
| --- | --- |
|  | ToggleSwitch toggleSwitch2 = new ToggleSwitch();  toggleSwitch2.IsOn = true;  toggleSwitch2.OnContent = "Включено";  toggleSwitch2.OffContent = "Выключено";  toggleSwitch2.Header = "Переключатель"; |

Для вывода изображений используется элемент Image. Он работает со следующими форматами файлов: jpeg, png, bmp, tiff, gif, ico.

Чтобы установить отображаемое изображение Image использует свойство Source, которому присваивается путь к изображению. В качестве пути могут применяться адреса в интернете или пути к файлам внутри приложения.

Пути к изображению отсчитываются относительно корня проекта. Допустим, в корне проекта лежит графический файл river.png. Выведем его на страницу:

|  |  |
| --- | --- |
|  | <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">      <Image Source="river.png" />  </Grid> |



Мы можем управлять отображением картинки, используя свойство Stretch. Оно принимает следующие значения:

* None: отсутствие масштабирования - изображение отображается в натуральную длину и ширину. Если размер изображения превышает размер Image, то изображение усекается
* Uniform: изображение масштабируется так, чтобы наиболее эффективно вписаться в размеры Image без потери аспектного пространства (отношения длины изображения к ширине). Это значение по умолчанию.
* Fill: изображение масштабируется таким образом, чтобы заполнить все пространство Image
* UniformToFill: изображение масштабируется таким образом, чтобы заполнить все пространство Image с сохранением аспектного отношения

Выше был приведен пример с Stretch="Uniform". То же самое изображение при значении Stretch="Fill":



И значение Stretch="UniformToFill":



### Свойство Clip. Обрезка изображения

С помощью свойства Clip можно вырезать на элементе Image определенную часть изображения. Например:

|  |  |
| --- | --- |
|  | <Image Source="river.png" Stretch="Fill">      <Image.Clip>          <RectangleGeometry Rect="20,20,300,250" />      </Image.Clip>  </Image> |

Свойство Clip принимает объект RectangleGeometry, которое с помощью свойства Rect устанавливает вырезаемую область:

### Программное создание

Программное создание объекта Image:

|  |  |
| --- | --- |
|  | Image image = new Image();  Windows.UI.Xaml.Media.Imaging.BitmapImage bitmap = new Windows.UI.Xaml.Media.Imaging.BitmapImage();  bitmap.UriSource = new Uri("ms-[appx:///river.png](appx://river.png)");  image.Source = bitmap;  image.Stretch = Stretch.UniformToFill; |

CalendarDatePicker представляет календарь, в котором можно выбрать дату.

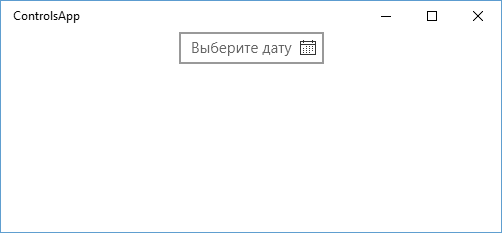
|  |  |
| --- | --- |
|  | <StackPanel HorizontalAlignment="Center">      <CalendarDatePicker x:Name="calendar" PlaceholderText="Выберите дату" CalendarIdentifier="GregorianCalendar" />  </StackPanel> |

Свойство PlaceholderText задает текст для ввода в календаре. По умолчанию равно строке "select a date".

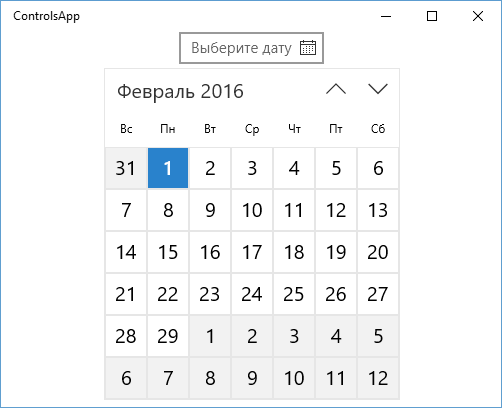
Свойство CalendarIdentifier задает тип календаря в зависимости от региональных предпочтений. Может принимать следующие значения:

* GregorianCalendar: григорианский календарь, значение по умолчанию
* HebrewCalendar: еврейский календарь
* HijriCalendar: мусульманский календарь
* JapaneseCalendar: японский календарь
* JulianCalendar: юлианский календарь
* KoreanCalendar: корейский календарь
* TaiwanCalendar: тайваньский календарь
* ThaiCalendar: тайский календарь
* UmAlQuraCalendar: другая версия мусульманского календаря

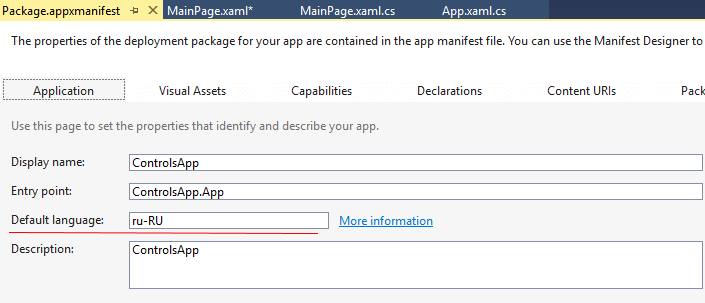
В итоге после запуска приложения мы увидим поле:



При нажатие на это поле отобразится календарик, в котором мы сможем установить дату:



Календарь использует языковые настройки проекта, которые по умолчанию настроены на применение англоязычной культуры. Поэтому, чтобы локализовать на нужную нам культуру, нам может потребоваться изменить язык приложения по умолчанию в файле манифеста:

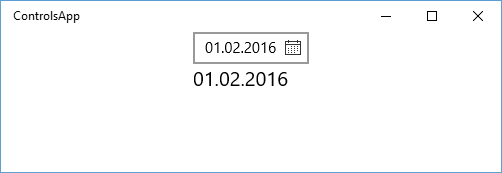


Для отслеживания изменения выбранной даты в календаре мы можем обрабатывать событие DateChanged:

|  |  |
| --- | --- |
|  | <StackPanel HorizontalAlignment="Center">      <CalendarDatePicker x:Name="calendar" PlaceholderText="Выберите дату" DateChanged="calendar\_DateChanged" />      <TextBlock x:Name="textBlock1" />  </StackPanel> |

В файле кода c# в обработчике выведем выбранную дату в текстовый блок:

|  |  |
| --- | --- |
|  | private void calendar\_DateChanged(CalendarDatePicker sender, CalendarDatePickerDateChangedEventArgs args)  {      DateTime selectedDate = calendar.Date.Value.DateTime;      textBlock1.Text = selectedDate.ToString("dd/MM/yyyy");        // также мы можем получить старую и новую дату таким образом      //DateTime? newDate = args.NewDate.Value.DateTime;      //DateTime? oldDate = args.OldDate.Value.DateTime;  } |



Еще пара свойств, которые мы можем использовать у календаря: DateFormat и DisplayMode.

DateFormat определяет формат отображения даты:

* {}{day.integer} {month.full} {year.full}: формат "1 февраля 2016"
* {}{day.integer}/{month.integer}/{year.full}: формат "1/2/2016"
* {}{month.full} {day.integer}, {year.full}: формат "февраль 1, 2016"
* {}{month.integer}/{day.integer}/{year.full}: формат "2/1/2016"
* {}{year.full}/{month.integer}/{day.integer}: формат "2016/2/1"

DisplayMode определяет формат диапазонов в календаре:

* Decade: календарь разделен по десятилетиям
* Month: разделение по месяцам
* Year: разделение по годам

Программное создание календаря в коде c#:

|  |  |
| --- | --- |
|  | CalendarDatePicker calendar = new CalendarDatePicker();  calendar.Date = DateTime.Now.Date;  calendar.DisplayMode = CalendarViewDisplayMode.Month;  calendar.PlaceholderText = "Выберите дату"; |

### CalendarView

CalendarView в многом похож на CalendarDatePicker, только представляет открытый календарь без текстового поля ввода. Главная его особенность - возможность выделения дат. Для настройки выделения дат используется множество разных свойств. Так, свойство SelectionMode устанавливает режим выделения дат и может принимать следующие значения:

* None: нельзя выделять даты
* Single: можно выделить только одну дату
* Multiple: можно выделять сразу несколько дат

Еще набор свойств устанавливает настройки цветов, шрифтов и т.д.: SelectedBorderBrush (цвет границы выделенной даты), SelectedForeground (цвет шрифта выделенной даты), SelectedHoverBorderBrush (цвет границы при наведении).

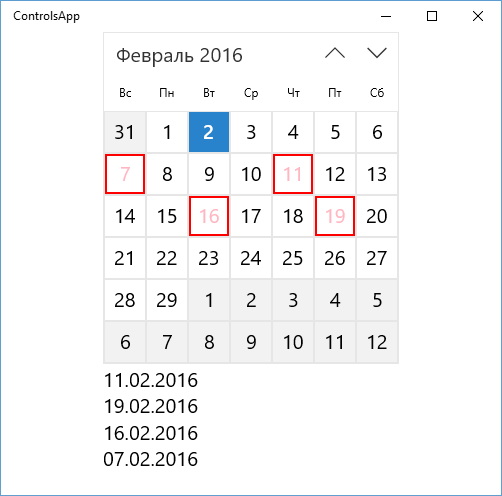
Еще ряд свойств устанавливают шрифты дат: DayItemFontFamily (семейство шрифтов дат), DayItemFontSize (размер шрифта дат) и т.д.

Через событие мы можем отследить выделение дат в CalendarView:

|  |  |
| --- | --- |
|  | <StackPanel HorizontalAlignment="Center" Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">      <CalendarView x:Name="calendarView" SelectionMode="Multiple" SelectedBorderBrush="Red"          SelectedForeground="LightPink"  SelectedDatesChanged="CalendarView\_SelectedDatesChanged" />      <TextBlock x:Name="textBlock1" />  </StackPanel> |

И обработчик в файле кода:

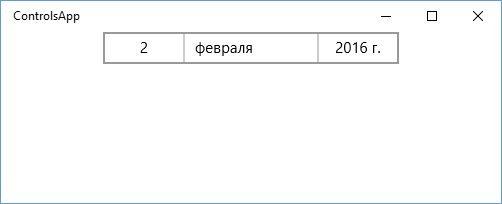
|  |  |
| --- | --- |
|  | private void CalendarView\_SelectedDatesChanged(CalendarView sender, CalendarViewSelectedDatesChangedEventArgs args)  {      textBlock1.Text = "";        foreach (var d in calendarView.SelectedDates)          textBlock1.Text += d.ToString("dd/MM/yyyy") + "\n";        // args.AddedDates - новые выделенные даты      //args.RemovedDates - даты с которых сняты выделения  } |



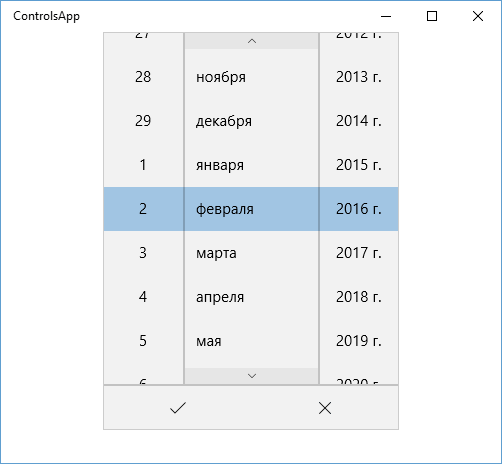
### DatePicker

Элемент DatePicker используется для выбора даты:

|  |  |
| --- | --- |
|  | <DatePicker x:Name="datePicker1" /> |



При нажатии на элемент появляется список для выбора даты, месяца, года:



Среди свойств DatePicker можно отметить следующие:

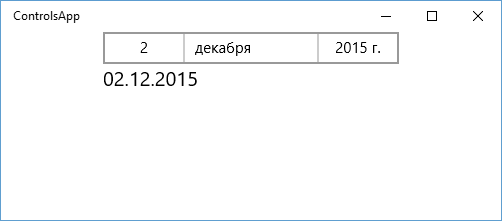
* DayFormat: формат отображения дней
* MonthFormat: формат отображения месяцев
* YearFormat: формат отображения годов
* CalendarIdentifier: идентификатор календаря, который имеет те же самые значения, что и у CalendarDatePicker
* MaxYear: максимальный год для выбора
* MinYear: наименьший год для выбора
* Header: заголовок над элементом управления

Для обработки выбора даты мы можем использовать событие DateChanged:

|  |  |
| --- | --- |
|  | <StackPanel HorizontalAlignment="Center">      <DatePicker x:Name="datePicker1" DateChanged="datePicker\_DateChanged" />      <TextBlock x:Name="textBlock1" />  </StackPanel> |

И обработчик в файле кода C#:

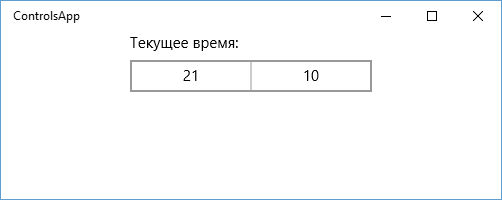
|  |  |
| --- | --- |
|  | private void datePicker\_DateChanged(object sender, DatePickerValueChangedEventArgs e)  {      DateTimeOffset dateOffset = datePicker1.Date;      textBlock1.Text = dateOffset.Date.ToString("dd.MM.yyyy");  } |



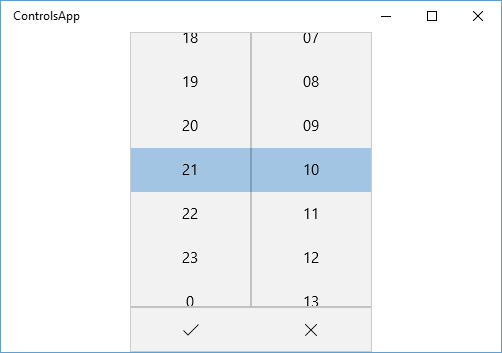
### TimePicker

TimePicker применяется для отображения или установки времени:

|  |  |
| --- | --- |
|  | <TimePicker HorizontalAlignment="Center" VerticalAlignment="Top"              Header="Текущее время:" ClockIdentifier="24HourClock" /> |



Свойство Header задает заголовок, а свойство ClockIdentifier формат отображения времени. Оно принимает два значения: 12HourClock (12-часовой формат) и 24HourClock (24-часовой формат). При нажатии на часы или минуты произойдет открытие списка, в котором можно установить новое время:



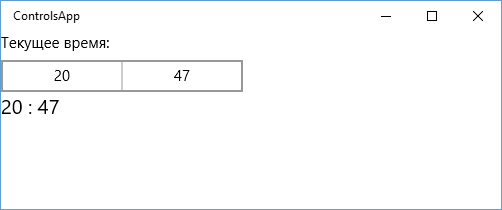
Чтобы отследить изменение времени в TimePicker мы можем обрабатывать событие TimeChanged:

|  |  |
| --- | --- |
|  | <StackPanel>      <TimePicker x:Name="timePicker1" Header="Текущее время:" ClockIdentifier="24HourClock" TimeChanged="TimePicker\_TimeChanged" />      <TextBlock x:Name="textBlock1" />  </StackPanel> |

В обработчике будем выводить выбранное время в текстовый блок:

|  |  |
| --- | --- |
|  | private void TimePicker\_TimeChanged(object sender, TimePickerValueChangedEventArgs e)  {      TimeSpan time = timePicker1.Time;      textBlock1.Text = $"{time.Hours} : {time.Minutes}";  } |

Свойство Time хранит выбранное время в виде объекта TimeSpan. Получив этот объект, мы можем использовать его свойства и методы:



Элемент Flyout аналогичен контекстным меню, всплывающим подсказкам или диалоговым окнам. Данный элемент используется вместе с другими элементами, для которых надо создать контекстное меню. Так, для всех кнопок его можно прикрепить через свойство Button.Flyout, а для всех остальных элементов используется свойство FlyoutBase.AttachedFlyout.

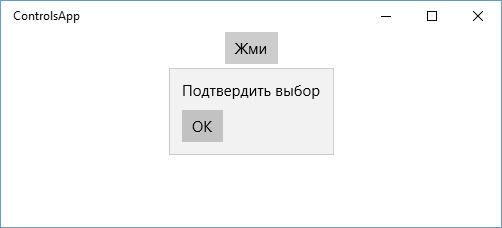
Если к кнопке прикрепить Flyout, то он отобразится после нажатия на кнопку. Например:

|  |  |
| --- | --- |
|  | <Button Content="Жми">      <Button.Flyout>          <Flyout x:Name="dialogBox">              <Flyout.Content>                  <StackPanel>                      <TextBlock Text="Подтвердить выбор" />                      <Button Click="Button\_Click" Content="OK" Margin="0 10 0 0" />                  </StackPanel>              </Flyout.Content>          </Flyout>      </Button.Flyout>  </Button> |

Здесь Flyout фактически представляет диалоговое окно с текстовой надписью и копкой. В коде C# мы можем задать обработчик для кнопки такого диалогового окна:

|  |  |
| --- | --- |
|  | private void Button\_Click(object sender, RoutedEventArgs e)  {      dialogBox.Hide(); // закрываем окно  } |

В данном случае просто закрываем окно. В итоге при нажатии на копку нам отобразится следующее диалоговое окно:



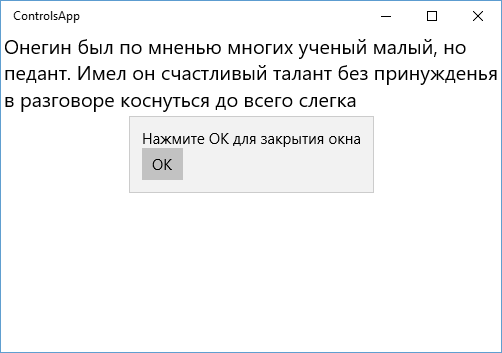
Выше мы рассмотрели установку Flyout для кнопок, однако для других элементов управления все будет немного по другому. Так, Flyout надо программно отображать (сам по себе он не отобразится) с помощью метода ShowAt() или ShowAttachedFlyout. А для прикрепления Flyout к другим элементам управления, отличным от кнопок, применяется свойство FlyoutBase.AttachedFlyout

Посмотрим на примере TextBlock:

|  |  |
| --- | --- |
|  | <TextBlock Tapped="TextBlock\_Tapped" TextWrapping="Wrap" Text="Онегин был по мненью многих...">      <FlyoutBase.AttachedFlyout>          <Flyout x:Name="dialogBox">              <StackPanel>                  <TextBlock Text="Нажмите ОК для закрытия окна" />                  <Button Content="OK" Click="Exit\_Click" />              </StackPanel>          </Flyout>      </FlyoutBase.AttachedFlyout>  </TextBlock> |

И в файле кода зададим обработчики, которые будут управлять поведением Flyout:

|  |  |
| --- | --- |
|  | private void Exit\_Click(object sender, RoutedEventArgs e)  {      dialogBox.Hide();  }    private void TextBlock\_Tapped(object sender, TappedRoutedEventArgs e)  {      dialogBox.ShowAt((TextBlock)sender);  } |



### MenuFlyout

На основе Flyout существует несколько дополнительных элементов управления: DatePickerFlyout (выпадающее окно для выбора даты), TimePickerFlyout и MenuFlyout (контекстное меню).

Применяются они аналогично. Например, используем MenuFlyout:

|  |  |
| --- | --- |
|  | <Button Content="Жми">      <Button.Flyout>          <MenuFlyout Placement="Bottom">              <MenuFlyoutSubItem Text="Проект">                  <MenuFlyoutItem Text="Создать" />                  <MenuFlyoutItem Text="Открыть" />              </MenuFlyoutSubItem>              <MenuFlyoutItem Text="Правка" />              <MenuFlyoutSeparator />              <ToggleMenuFlyoutItem IsChecked="True" Text="Сохранить" />              <MenuFlyoutItem Text="Выход" Click="Exit\_Click" />          </MenuFlyout>      </Button.Flyout>  </Button> |

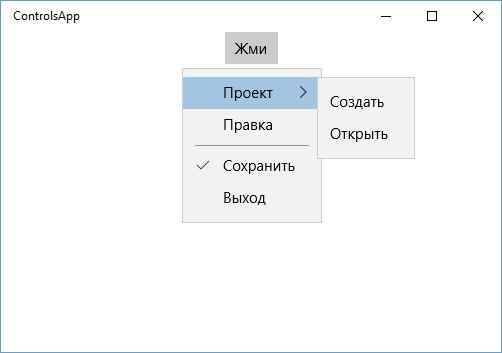
Внутрь MenuFlyout добавляются отдельные пункты меню. Если нам нужен одиночный пункт меню, то применяется элемент MenuFlyoutItem. Если мы хотим создать иерархической меню, в котором есть подменю, то используется элемент MenuFlyoutSubItem, в который добавляются элементы подменю.

Для красоты можно использовать элемент-разграничитель в виде черты - MenuFlyoutSeparator. И кроме того, если нам нужен пункт меню, который может находиться в отмеченном и неотмеченном состоянии, то мы можем воспользоваться элементом ToggleMenuFlyoutItem.

К каждому пункту меню можно прикрепить обработчики событий. Так, в примере выше для последнего пункта задан обработчик события нажатия. Определим его в файле кода c#:

|  |  |
| --- | --- |
|  | private void Exit\_Click(object sender, RoutedEventArgs e)  {      App.Current.Exit(); // выход из приложения  } |

В итоге при нажатии на кнопку нам отобразится вот такое контекстное меню:



Среди собственной функциональности MenuFlyout следует отметить разве что свойство Placement, которое задает положение контекстного меню и может принимать следующие значения:

* Bottom: расположение под кнопкой
* Fill: меню растягивается на полную ширину
* Top: расположение над кнопкой
* Left: расположение слева
* Right: расположение справа

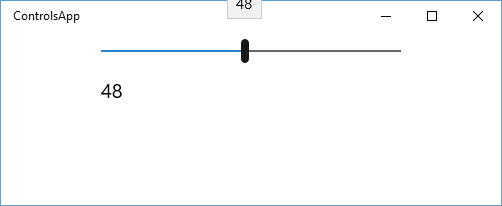
Slider представляет собой ползунок с делениями:

|  |  |
| --- | --- |
|  | <StackPanel HorizontalAlignment="Center">      <Slider x:Name="slider" Minimum="0" Maximum="100" Value="20" Orientation="Horizontal" Width="300"          ValueChanged="slider\_ValueChanged" />      <TextBlock x:Name="textBlock1" />  </StackPanel> |

Слайдер позволяет задать минимальное и максимальное значения с помощью свойств Maximum и Minimum. Value задает начальное значение. А свойство Orientation указывает на ориентацию слайдера - горизонтальную или вертикальную.

Событие ValueChanged позволяет отреагировать на изменение значения слайдера. Определим в файле кода для него обработчик:

|  |  |
| --- | --- |
|  | private void slider\_ValueChanged(object sender, RangeBaseValueChangedEventArgs e)  {      if (textBlock1 != null)          textBlock1.Text = slider.Value.ToString();  } |

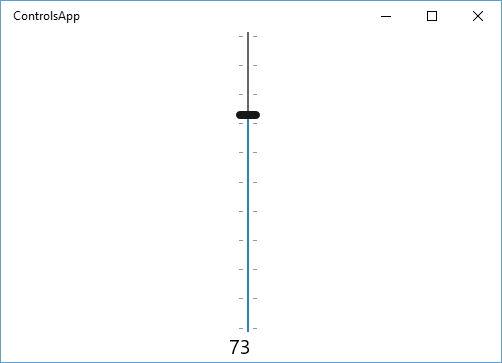


Также можно выделить свойства: TickFrequency, которое указывает на количество делений на слайдере, и TickPlacement - положение делений, которое может принимать следующие значения:

* BottomRight: деления располагаются внизу (при горизонтальной ориентации) и справа (при вертикальной)
* TopLeft: деления располагаются вверху (при горизонтальной ориентации) и слева (при вертикальной)
* Inline: деления встроены в слайдер
* Outside: деления по обе стороны слайдера
* None: отсутствие делений

Например, применим эти свойства:

|  |  |
| --- | --- |
|  | <Slider x:Name="slider" Minimum="0" Maximum="100" Value="20" Orientation="Vertical" Height="300"      TickFrequency="10" TickPlacement="Outside" ValueChanged="slider\_ValueChanged" /> |



**Задание лабораторной работы**

Вариант: 20

Вычислить и вывести сумму элементов матрицы A(12,12), расположенных над главной диагональю матрицы.

**Скриншоты лабораторной работы**

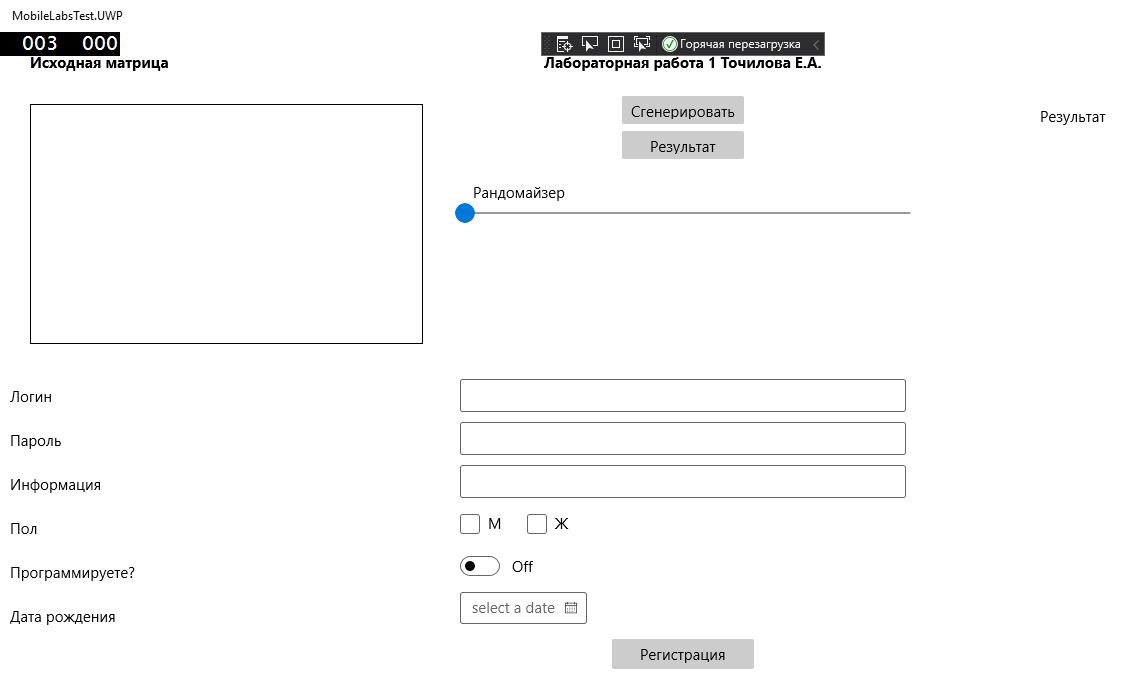
****

Рисунок . Работа программы 1

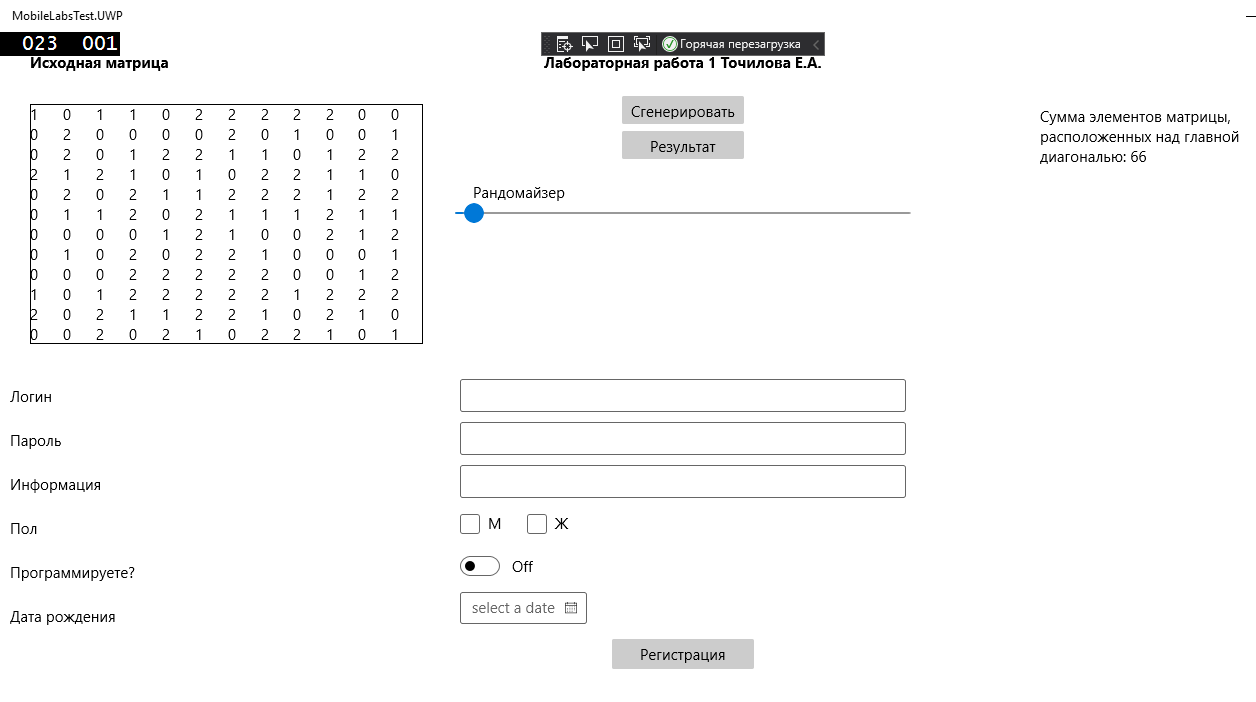
****

Рисунок . Работа программы 2

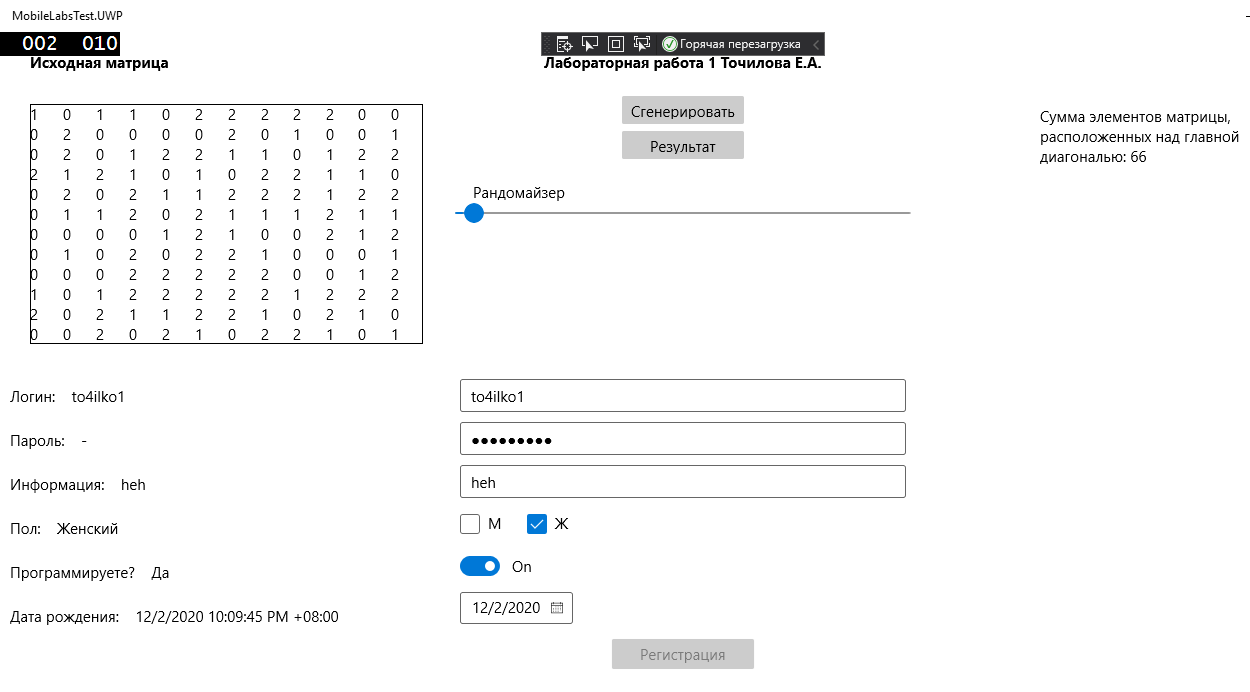
****

Рисунок . Работа программы 3

**Код программы MainPage.xaml.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.InteropServices.WindowsRuntime;

using Windows.Foundation;

using Windows.Foundation.Collections;

using Windows.UI.Xaml;

using Windows.UI.Xaml.Controls;

using Windows.UI.Xaml.Controls.Primitives;

using Windows.UI.Xaml.Data;

using Windows.UI.Xaml.Input;

using Windows.UI.Xaml.Media;

using Windows.UI.Xaml.Navigation;

namespace MobileLabsTest.UWP

{

public sealed partial class MainPage

{

public TextBlock[,] textBlocks = new TextBlock[12,12];

public TextBlock resultTextBlock = new TextBlock();

public MainPage()

{

this.InitializeComponent();

//LoadApplication(new MobileLabsTest.App());

Random random = new Random();

for (int i = 0; i < 12; i++)

{

for (int j = 0; j < 12; j++)

{

TextBlock rowtext = new TextBlock();

Grid.SetColumn(rowtext, i);

Grid.SetRow(rowtext, j);

matrix.Children.Add(rowtext);

textBlocks[i,j] = rowtext;

}

}

}

public int[,] elements = new int[12, 12];

public int next = 0;

private void Button\_Click(object sender, RoutedEventArgs e)

{

//Заполняем табличку рандомными числами

Random random = new Random();

for (int i = 0; i < 12; i++)

{

for (int j = 0; j < 12; j++)

{

textBlocks[i, j].Text = Convert.ToString(random.Next(next));

elements[i, j] = Convert.ToInt32(textBlocks[i, j].Text);

}

}

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

int s = 0;

for (int i = 1; i < 12; i++)

{

for (int j = 1; j < 12; j++)

{

if (j > i)

s += elements[i,j];

}

}

result.Text = "Сумма элементов матрицы, расположенных над главной диагональю:" + " " + Convert.ToString(s);

}

public void ClearResult()

{

result.Text = "";

}

private void randomSlider\_ValueChanged(object sender, RangeBaseValueChangedEventArgs e)

{

next = Convert.ToInt32(randomSlider.Value);

}

private void Registration\_Click(object sender, RoutedEventArgs e)

{

loginText.Text = "Логин: " + login.Text;

passText.Text = "Пароль: -";

informText.Text = "Информация: " + info.Text;

if (female.IsChecked == true)

{

sexText.Text = "Пол: Женский";

}

else

{

sexText.Text = "Пол: Мужской";

}

if (prog.IsOn == true)

{

progrText.Text = "Программируете? Да";

}

else

{

progrText.Text = "Программируете? Нет";

}

birthdayText.Text = "Дата рождения: " + Convert.ToString(date.Date);

Registration.IsEnabled = false;

}

private void male\_Checked(object sender, RoutedEventArgs e)

{

female.IsChecked = false;

}

private void female\_Checked(object sender, RoutedEventArgs e)

{

male.IsChecked = false;

}

}

}

**Код программы MainPage.xaml**

<forms:WindowsPage

x:Class="MobileLabsTest.UWP.MainPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:forms="using:Xamarin.Forms.Platform.UWP"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="using:MobileLabsTest.UWP"

Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}" RequestedTheme="Light">

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<TextBlock FontWeight="Bold" TextWrapping="Wrap" VerticalAlignment="Top" HorizontalAlignment="Center" Text="Лабораторная работа 1 Точилова Е.А." Height="38" Margin="0,20,0,20" Grid.Column="1"/>

<TextBlock FontWeight="Bold" TextWrapping="Wrap" VerticalAlignment="Top" HorizontalAlignment="Left" Text="Исходная матрица" Height="38" Width="200" Margin="30,20,0,0"/>

<Button x:Name="genButton" Grid.Row="0" Grid.Column="1" VerticalAlignment="Top" HorizontalAlignment="Center" Margin="0,63,0,0" Width="124" Height="30" Click="Button\_Click" Content="Сгенерировать"></Button>

<Button x:Name="resultButton" Grid.Row="0" Grid.Column="1" VerticalAlignment="Top" HorizontalAlignment="Center" Margin="0,98,0,0" Width="124" Height="30" Click="Button\_Click\_1" Content="Результат">

<Button.Flyout>

<Flyout>

<TextBlock Text="Результат вычислений представлен справа!"/>

</Flyout>

</Button.Flyout>

</Button>

<TextBlock Text="Рандомайзер" Margin="18,150,16,54" Grid.Column="1"/>

<Slider Grid.Row="0" Grid.Column="1" x:Name="randomSlider"

Minimum="1"

Maximum="99" Margin="0,165,0,10" ValueChanged="randomSlider\_ValueChanged" />

<Grid Margin="30,72,32,30" Grid.Column="0" Grid.Row="0" x:Name="matrix">

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Border BorderThickness="1" BorderBrush="Black" Grid.ColumnSpan="12" Grid.RowSpan="12"/>

</Grid>

<TextBlock x:Name="result" TextWrapping="Wrap" Text="Результат" Width="200px" Height="200px" Margin="18,60,16,54" Grid.Column="2" Grid.Row="0"/>

<Grid Grid.Column="0" Grid.Row="1">

<Grid.ColumnDefinitions>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<TextBlock x:Name="loginText" Margin="10 12 0 12" Grid.Row="0" Text="Логин"/>

<TextBlock x:Name="passText" Margin="10 12 0 12" Grid.Row="1" Text="Пароль"/>

<TextBlock x:Name="informText" Margin="10 12 0 12" Grid.Row="2" Text="Информация"/>

<TextBlock x:Name="sexText" Margin="10 12 0 12" Grid.Row="3" Text="Пол"/>

<TextBlock x:Name="progrText" Margin="10 12 0 12" Grid.Row="4" Text="Программируете?"/>

<TextBlock x:Name="birthdayText" Margin="10 12 0 12" Grid.Row="5" Text="Дата рождения"/>

</Grid>

<Grid Grid.Column="1" Grid.Row="1">

<Grid.ColumnDefinitions>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<TextBox x:Name="login" Grid.Row="0" Margin="5"/>

<PasswordBox x:Name="password" Grid.Row="1" Margin="5"/>

<TextBox x:Name="info" Grid.Row="2" Margin="5"/>

<CheckBox x:Name="male" Margin="5" Content="М" Grid.Row="3" Width="72" Checked="male\_Checked"/>

<CheckBox x:Name="female" Content="Ж" Grid.Row="3" Checked="female\_Checked" Margin="72 5 5 5"/>

<ToggleSwitch x:Name="prog" Grid.Row="4" Margin="5"/>

<CalendarDatePicker x:Name="date" Grid.Row="5" Margin="5"/>

</Grid>

<Grid Grid.Column="2" Grid.Row="1">

<Grid.ColumnDefinitions>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

</Grid>

<Button x:Name="Registration" Content="Регистрация" Width="144" HorizontalAlignment="Center" Click="Registration\_Click" Grid.Column="0" Grid.Row="2" Grid.ColumnSpan="3" VerticalAlignment="Top"/>

</Grid>

</forms:WindowsPage>