Пользовательский интерфейс: работа с XAML для формирования пользовательского интерфейса

Разработка пользовательского интерфейса является важной частью процесса разработки программного приложения.

Пользовательский интерфейс для мобильных приложений может быть разработан как с использованием программного кода на языке программирования (например, Си Шарп, Java, Kotlin, Swift), так и с использованием языка разметки XAML

Программный код XAML имеет несколько преимуществ по сравнению с эквивалентным программным кодом, например, на языке Си Шарп

- XAML является более лаконичным и удобочитаемым, чем эквивалентный программный код на языке Си Шарп.
- Более наглядное отображение иерархии "родители-потомки" объектов пользовательского интерфейса.

Существуют также недостатки, в основном связанные с ограничениями, встроенными в языки разметки:

- Программный код XAML не может содержать код обработчиков событий: все обработчики событий должны быть определены в файле программного кода на языке Си Шарп который имеет расширение xaml.cs.
- Программный код XAML не может содержать циклы для повторяющейся обработки данных
- Программный код XAML не может содержать условные операторы
- Программный код XAML обычно не может создавать объекты (экземпляры классов) с использованием конструкторов
- Программный код XAML не может вызывать методы классов

Файл с расширением xaml содержит разметку визуального интерфейса страницы и представляет собой файл xml. Первой строкой идет стандартное определение xml-файла.

В определении корневого элемента ContentPage подключаются четыре пространства имен с помощью атрибутов xmlns.

Пространство имен http://xamarin.com/schemas/2014/forms определяет большинство типов из Xamarin Forms, которые применяются для построения графического интерфейса.

Второе пространство имен http://schemas.microsoft.com/winfx/2009/xaml определяет типы XAML и типы для общеязыковой среды выполнения. Так как только одно пространство имен может быть базовым, то это пространство используется с префиксом x: xmlns:x.

Это значит, что те свойства элементов, которые заключены в этом пространстве имен, будут использоваться с префиксом x - x:Name или x:Class.

Это основные пространства имен.

Следующие два пространства имен используются только в процессе разработки в Visual Studio.

Выражение mc:Ignorable="d" говорит о том, что все элементы с префиксом d в процессе выполнения приложения будут игнорироваться и будет использоваться только в процессе разработки.

После подключения пространств имен идет атрибут x:Class="НазваниеПроекта.MainPage", который указывает на класс, представляющий данную страницу в пространстве имен «НазваниеПроекта».

Таким образом, в XAML предлагается простая схема определения составляющих пользовательского интерфейса и их свойств.

Каждый элемент должен иметь открытый и закрытый тег, как, например, у рассмотренного ранее элемента управления Label.

Тэг открывается с помощью открывающейся угловой скобки, а закрывается с помощью слэша и закрывающейся угловой скобки.

```
<Label
    Text="Xamarin.Forms"              HorizontalOptions="Center"
    VerticalOptions="StartAndExpand"  BackgroundColor="#EEEA0C"
    FontSize="30"                     TextColor="#000000"
    HeightRequest="50" />
```

В XAML свойства классов обычно устанавливаются в виде XML-атрибутов:

```
<Label Text="Hello, XAML!"
    VerticalOptions="Center"
    FontAttributes="Bold"
    FontSize="Large"
    TextColor="Aqua"

<Label
        Text="Пользовательский интерфейс"
        HorizontalOptions="Center"   VerticalOptions="StartAndExpand"
        BackgroundColor="#0F29E9"   FontSize="20"
        FontAttributes="Bold"          TextColor="#E3DE0C"
        HeightRequest="40" />
```

Существует альтернативный способ задания свойства в XAML.

Для этого необходимо в программный код добавить открывающий и закрывающий теги, состоящие из имени класса и имени свойства, разделенных точкой. При этом синтаксис элемента свойства можно использовать для нескольких свойств:

```
<Label Text="Hello, XAML!"
     VerticalOptions="Center">
   <Label.FontAttributes>
      Bold
   </Label.FontAttributes>
   <Label.FontSize>
      Large
   </Label.FontSize>
</Label>
```

Каждый элемент в XAML представляет объект определенного класса Си Шарп, а атрибуты элементов соотносятся со свойствами этих классом.

Например, элемент ContentPage фактически будет представлять объект одноименного класса ContentPage, а элемент Label - объект класса Label.

Отметим, что свойства классов могут принимать значения различных типов: string, double, int и т.д.

В программном коде XAML соответствующие атрибуты элементов имеют текстовые значения.

```
<Label
      Text="Пользовательский интерфейс"
      HorizontalOptions="Center"   VerticalOptions="StartAndExpand"
      BackgroundColor="#0F29E9"   FontSize="20"
      FontAttributes="Bold"          TextColor="#E3DE0C"
      HeightRequest="40" />
```

В программном коде XAML могут быть использованы присоединенные свойства, которые в XAML-файлах распознаются как атрибуты, содержащие класс и имя свойства, разделенные точкой.

Они называются присоединенными свойствами, так как они определены одним классом, но присоединены к другим объектам.

Задание 1

Опишем использование присоединенного свойства на примере макета AbsoluteLayout.

Создадим проект Chess для мультиплатформенного программного приложения Xamarin.Forms с использованием шаблона «Пустой».

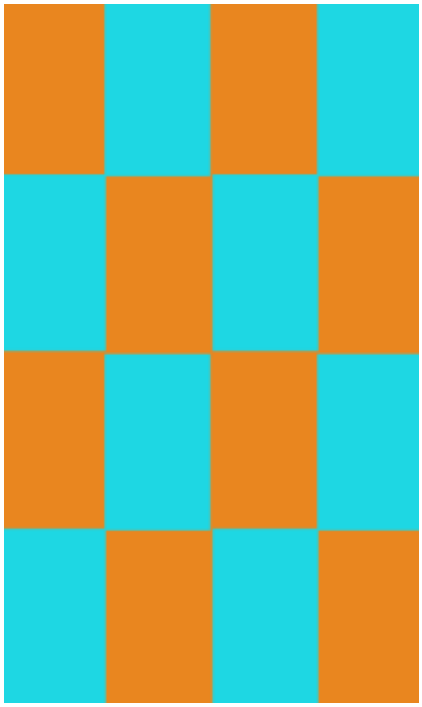
Откроем файл MainPage.xaml и наберем там программный код.

В программном коде с помощью макета AbsotutLayout и восьми элементов управления BoxView реализуется шахматная доска с использованием функций пропорционального позиционирования элементов управления BoxView.

В качестве присоединенных свойств для элементов управления BoxView используются свойства LayoutBounds и LayoutFlags макета AbsoluteLayout.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class=" Chess.MainPage"
        Title="Absolute Demo Page">

  <AbsoluteLayout BackgroundColor="#FF8080">
    <BoxView Color="#8080FF"
          AbsoluteLayout.LayoutBounds="0.33, 0, 0.25, 0.25"
          AbsoluteLayout.LayoutFlags="All" />

    <BoxView Color="#8080FF"
          AbsoluteLayout.LayoutBounds="1, 0, 0.25, 0.25"
          AbsoluteLayout.LayoutFlags="All" />

    <BoxView Color="#8080FF"
          AbsoluteLayout.LayoutBounds="0, 0.33, 0.25, 0.25"
          AbsoluteLayout.LayoutFlags="All" />

    <BoxView Color="#8080FF"
          AbsoluteLayout.LayoutBounds="0.67, 0.33, 0.25, 0.25"
          AbsoluteLayout.LayoutFlags="All" />

    <BoxView Color="#8080FF"
          AbsoluteLayout.LayoutBounds="0.33, 0.67, 0.25, 0.25"
          AbsoluteLayout.LayoutFlags="All" />

    <BoxView Color="#8080FF"
          AbsoluteLayout.LayoutBounds="1, 0.67, 0.25, 0.25"
          AbsoluteLayout.LayoutFlags="All" />

    <BoxView Color="#8080FF"
          AbsoluteLayout.LayoutBounds="0, 1, 0.25, 0.25"
          AbsoluteLayout.LayoutFlags="All" />

    <BoxView Color="#8080FF"
          AbsoluteLayout.LayoutBounds="0.67, 1, 0.25, 0.25"
          AbsoluteLayout.LayoutFlags="All" />
  </AbsoluteLayout>
</ContentPage>
```

При запуске программного приложения отображается пользовательский интерфейс, соответствующий набранному программному году XAML



В программных приложениях для того, чтобы избежать наложения отображаемой страницы на строку состояния на экране мобильного устройства в случае использования платформы iOS, устанавливается свойство Padding для страницы в программном коде XAML.

Также для учета платформы, на которой запускается программное приложение, используется класс OnPlatform для работы со свойством Padding.

Класс OnPlatform имеет свойство Platforms. Элементы On устанавливают значения свойства Platform, а также значение свойства Value для задания Thickness Для этого задается значение свойства Thickness с помощью атрибута x:TypeArguments.

Значения для параметра Thickness определяют ширину пространства для свойства Padding.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         x:Class="...">

  <ContentPage.Padding>
    <OnPlatform x:TypeArguments="Thickness">
      <OnPlatform.Platforms>
        <On Platform="iOS" Value="0, 20, 0, 0" />
        <On Platform="Android" Value="0, 0, 0, 0" />
      </OnPlatform.Platforms>
    </OnPlatform>
  </ContentPage.Padding>
...
```

</ContentPage>

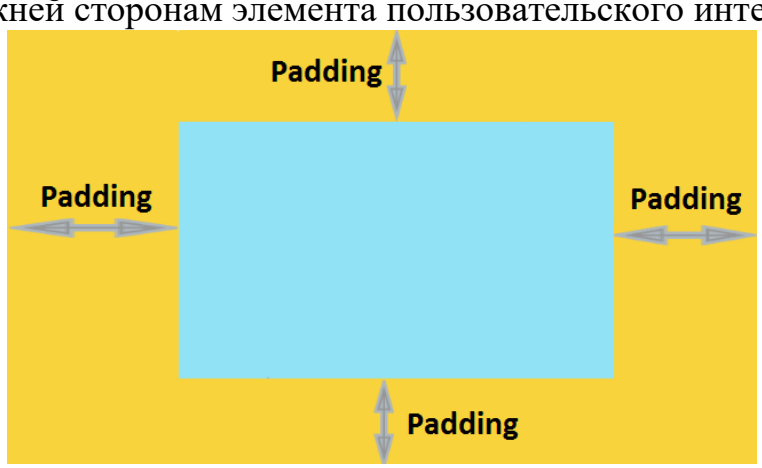
```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="...">

  <ContentPage.Padding>
    <OnPlatform x:TypeArguments="Thickness">
      <OnPlatform.Platforms>
        <On Platform="iOS" Value="0, 20, 0, 0" />
        <On Platform="Android" Value="0, 0, 0, 0" />
      </OnPlatform.Platforms>
    </OnPlatform>
  </ContentPage.Padding>
 ...
</ContentPage>
```

Свойство Padding задает размер свободного пространства на границах элемента пользовательского интерфейса. Для задания свойства Padding могут быть использованы три варианта Thickness,

Thickness с одним параметром применяется к левой, верхней, правой и нижней сторонам элемента пользовательского интерфейса;

Thickness с двумя (горизонтальным и вертикальным) значениями применяется соответственно к левой и правой, и к верхней и нижней сторонам элемента пользовательского интерфейса;

Thickness с четырьмя значениями применяются к левой, верхней, правой и нижней сторонам элемента пользовательского интерфейса.



Задание 2

В программном коде XAML часто применяются расширения разметки, которые позволяют задавать свойства объектов, на которые косвенно ссылаются другие объекты в программном коде. Таким образом, расширения разметки XAML предназначены для совместного использования объектов в программном коде.

Рассмотрим использование расширения разметки.

Для этого создадим проект MarkExt для мультиплатформенного приложения Xamarin.Forms с использованием шаблона «Пустой». Откроем файл MainPage.xaml и наберем там программный код.

В программном коде для реализации расширений разметки применяется словарь ресурсов (ResourceDictionary) свойства Resources страницы ContentPage для хранения значений свойств для совместного использования посредством ссылок на них из программного кода XAML.

Чтобы использовать словарь ресурсов на странице в программном коде используется пара тегов свойства Resources страницы ContentPage. Для задания словаря ресурсов используются теги ResourceDictionary.

В словарь ресурсов добавляются элементы пользовательского интерфейса и значения свойств. Для каждого элемента задается ключ словаря, заданный с помощью атрибута x:Key.

Задаются два элемента, являющиеся объектами типа LayoutOptions, каждый из которых имеет уникальный ключ и набор свойств.

С помощью атрибута x:Key задается элемент словаря borderWidth (для задания толщины границ элементов управления), который является значением типа double и значение которого равно 3.

С помощью атрибута x:Key задается элемент словаря rotationAngle (для задания угла поворота элемента управления), который является значением типа double и значение которого равно -15 градусам.

С помощью атрибута x:Key задается элемент словаря для свойства fontSize (для задания размера шрифта), которое является значением типа double и значение которого равно 30.

С помощью атрибута x:Key задается элемент словаря для свойства textColor (цвет страницы в зависимости от платформы, на которой запускается программное приложение). Для этого в словаре ресурсов используется класс OnPlatform для определения значения свойства textColor для различных платформ.

В пользовательском интерфейсе внутри страницы типа ContentPage находится макет типа StackLayout, внутри которого находятся два элемента управления Button.

Для задания значений свойств элементов управления производится ссылка на шесть объектов из словаря ресурсов с помощью расширения разметки StaticResource. Расширение разметки StaticResource отделяется фигурными скобками и указывает на объект из словаря.

Расширение разметки StaticResource применяется для доступа к объектам из словаря только один раз при создании элементов управления на странице.

Расширение разметки DynamicResource предназначено для доступа к объектам словаря, значения которых могут изменяться во время выполнения программного приложения.

```
ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="MarkExt.MainPage"
        Title="Расширение разметки"
        BackgroundColor="Bisque">


  <ContentPage.Resources>
    <ResourceDictionary>
      <LayoutOptions x:Key="horzOptions"
                Alignment="Center" />

      <LayoutOptions x:Key="vertOptions"
                Alignment="Center"
                Expands="True" />

      <x:Double x:Key="borderWidth">3</x:Double>

      <x:Double x:Key="rotationAngle">-15</x:Double>

      <x:Double x:Key="fontSize">24</x:Double>

      <OnPlatform x:Key="textColor"
              x:TypeArguments="Color">
        <On Platform="iOS" Value="Red" />
        <On Platform="Android" Value="Aqua" />
        <On Platform="UWP" Value="#80FF80" />
      </OnPlatform>

    </ResourceDictionary>
  </ContentPage.Resources>

  <StackLayout>
    <Button Text="Расширения XAML"
        WidthRequest="200"  BackgroundColor="Coral"
```

```
            HeightRequest="100" BorderColor="Black"

            HorizontalOptions="{StaticResource horzOptions}"
            VerticalOptions="{StaticResource vertOptions}"
            BorderWidth="{StaticResource borderWidth}"
            Rotation="{StaticResource rotationAngle}"
            TextColor="{StaticResource textColor}"
            FontSize="{StaticResource fontSize}" />

        <Button Text="Словарь ресурсов"
            WidthRequest="200"   BackgroundColor="Brown"
            HeightRequest="100"  BorderColor="Yellow"

            HorizontalOptions="{StaticResource horzOptions}"
            VerticalOptions="{StaticResource vertOptions}"
            BorderWidth="{StaticResource borderWidth}"
            Rotation="{StaticResource rotationAngle}"
            TextColor="{StaticResource textColor}"
            FontSize="{StaticResource fontSize}" />

    </StackLayout>
</ContentPage>
```

При запуске программного приложения отображается пользовательский интерфейс, соответствующий набранному программному году XAML.

Задание 3

Привязки данных позволяют так связывать свойства двух объектов, чтобы изменение одного из них привело к изменению другого. Привязки данных соединяют свойства двух элементов управления, которые называются источником данных и целевым объектом.

В программном коде XAML требуются два шага для привязки данных:

1. Назначение источника данных для целевого элемента управления объекта с помощью свойства BindingContext. Задание BindingContext целевого объекта производится с помощью расширения разметки в виде аргумента x:Reference.

2. Использование класса Binding для привязки свойства исходного объекта к свойству целевого объекта.

Рассмотрим использование привязки данных с помощью XAML.

Для этого создадим проект DatBind для мультиплатформенного приложения Xamarin.Forms с использованием шаблона «Пустой». Откроем файл MainPage.xaml и наберем там программный код.

Программный код XAML соответствует странице типа ContentPage, в которой содержится макет типа StackLayout, внутри которого расположены элемент управления Slider и два элемента управления Label. Первый элемент управления Label поворачивается на угол, который выбран с помощью элемента управления Slider. В другом элементе управления Label отображается значение угла, выбранного с помощью элемента управления Slider.

Описание элемента управления Slider содержит атрибут x:Name, на который в программном коде ссылаются два элемента управления Label с помощью расширения разметки x:Reference.

Расширение разметки x:Reference определяет имя источника данных, которое присваивается с использованием атрибута x:Name. Таким образом, в качестве источника данных выступает slider.

В программном коде класс Binding используется для привязки свойства Rotation для первого элемента управления Label с названием «ROTATION» к свойству Value элемента управления Slider, а также для привязки свойства Text второго элемента управления Label к свойству Value элемента управления Slider.

```
ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class=" DatBind.MainPage"
        Title="Slider Bindings"
        BackgroundColor="Orange">

  <StackLayout>
    <Label Text="ROTATION"
       VerticalTextAlignment="Center"
```

```
            HorizontalTextAlignment="Center"
            WidthRequest="200"   BackgroundColor="Brown"
            HeightRequest="80"  TextColor="Yellow"
            BindingContext="{x:Reference Name=slider}"
            Rotation="{Binding Path=Value}"
            FontAttributes="Bold"
            FontSize="30"
            HorizontalOptions="Center"
            VerticalOptions="CenterAndExpand" />

        <Slider x:Name="slider"
             Maximum="360"
             VerticalOptions="CenterAndExpand" />
        <Label BindingContext="{x:Reference slider}"
            VerticalTextAlignment="Center"
            HorizontalTextAlignment="Center"
            WidthRequest="400"   BackgroundColor="GreenYellow"
            HeightRequest="100"  TextColor="Red"
           Text="{Binding Value, StringFormat='Угол поворота  {0:F0} градусов'}"
            FontAttributes="Bold"
            FontSize="26"
            HorizontalOptions="Center"
            VerticalOptions="CenterAndExpand" />
    </StackLayout>
</ContentPage>
```

Запустим программное приложение. Продемонстрируем передачу данных от элемента управления Slider элементам управления Label