Tip Calculator

Portfolio Exercise

This exercise is to get you used to Xamarin and some of the standard controls. Unlike most other exercises, solutions are given at the end. However, you should use the Xamarin Standard Controls notes and see if you can work it out for yourself first. The more you work out now in this simple project, the better placed you'll be for the rest of the semester when solutions will not be given.

This portfolio exercise covers…

- Setting up a Xamarin project.
- Xamarin controls.
- State variables.
- Building a working app.

1. New Project

- Create a new project. The template you need is under 'Visual C#', then 'Cross Platform', then 'Mobile App (Xamarin.Forms)'.
    - There are three options - blank, master-detail and tabbed. Pick blank.
    - Make sure the code sharing strategy is '.NET Standard'. (Don't worry about what this means.)
    - Select all three platforms - Android, iOS and Windows.
- It will take a little while to generate your new solution.
- Once done, right click on the Solution in the Solution Explorer and select "Manage NuGet Packages"
    - Click the Update tab and update anything that needs it.
    - Visual Studio may want to restart.
- NuGet packages are bolt on libraries for various jobs, some of which are written by Microsoft and some by the Xamarin community. They need to be updated regularly and also at the beginning of any new project.
- Note that your Solution contains multiple projects. There should be one main project and one for each of the platforms you're supporting - Android, iOS, etc. Most of the work will be done in the main project but anything specific to a platform can be done in the project for that platform. You can delete any you don't want and add them back in again later if required.
- You can also change the platform you're testing for by right-clicking on the appropriate project and selecting 'Set as StartUpProject'. Do so for the UWP (Windows) platform. Since we're working on Windows, that is the easiest way to test.

A Quick but Important Aside

If you look in your code behind, you'll see this template code…

```
public MainPage()
{
      InitializeComponent();
}
```

InitializeComponent() is the method that loads the XAML and creates all the controls you put there. This is very important to remember because before InitializeComponent(), none of your UI exists yet. Any tweaks you want to do with your UI must be done after InitializeComponent().

2. Percentage Slider

- Create a horizontal stack layout. On the left side, insert a slider and name it "percentageSlider". Give it a decent width so you have room to slide it.
- The slider will be a percentage slider so give it a minimum of 0 and a maximum of 100.
- On the right, add a label called "percentageLabel" that says "0%"
- Set the starting value of the slider to 10%.
- Add an event handler for when the value of the slider changes. In the event handler, update the label to display the percentage. You'll need to display a percent sign after the number and the number should have no decimal places (no one is interested in tipping 10.25%, after all).
- Test the app to make sure it's working.

3. Tip Details

- Create a vertical stack view around the horizontal one you already have.
- Using either nested stack views or a grid view, create something similar to the layout below *above* the percentage slider. (You don't need the border - that's just so you can see the layout.)

Bill	$0.00
Tip	$0.00
Total	$0.00

- Give the dollar amount labels on the right the names billAmountLabel, tipAmountLabel and totalAmountLabel.
- In your code behind, add three floating point values: billAmount, tipAmount and totalAmount.
- Set the billAmount to 52.80, just for testing.

- Create a function called "UpdateUI" which calculates the tipAmount and totalAmount using the billAmount and the slider's value, and then updates all the appropriate labels in the user interface. Don't forget to put a dollar sign on the front.
- Call UpdateUI() from the slider's event handler.
- Test and confirm the value changes as the slider does and that you're getting the correct results (use a calculator).
- Set the billAmount variable to 0 once finished testing.

4. Calculator Buttons

It is common for calculator-type apps to use a custom keyboard that is on screen all the time, partially to save the time it takes for a mobile device keyboard to pop up on the screen, partially so the keyboard can be customised to have exactly what it needs and no more, and partially because otherwise you have a lot of empty screen space.

- In between the display of the various amounts and the slider, insert a new grid layout three rows wide and four columns tall. Each cell of the grid should contain a single button, like so…

7	8	9
4	5	6
1	2	3
0	.	C

- Set each button to call the *same event handler.* (We can tell which button is which from the text of each button.)
- In the event handler, write the button text to the console so you can test to see that it's working correctly with all the buttons.

5. Entering A Bill Amount

Entering numbers into the bill label using this custom keyboard has a little complexity. Here are the rules…

1. If the user presses "C", everything is reset.
2. If the user starts by typing a zero, nothing happens.
3. The user can only enter a decimal place once.
4. There can only be two digits after the decimal place.

You can keep track of this using state variables. If you haven't heard the term before, you've probably used them anyway. State variables don't' store data but rather keep track of what's going on. They are usually booleans or integers. For example, in this case, you would need something like…

- **hasStarted** - A boolean that tracks whether the user has pressed any key except C and 0. If they haven't, then pressing 0 does nothing. This handles rule 2 above.
- **hasTypedDecimal** - A boolean that tracks whether the user has typed a decimal point. If they have, they cannot type a second one. This handles rule 3 above.

- **numberOfDecimalDigits** - An integer that records how many numbers have been typed after the decimal point (so you need to check “hasTypedDecimal” first). If it is equal to 2, then no further numbers can be typed. This handles rule 4 above.

Write the code that allows the user to type in an amount of money so that it appears in the bill label. Don’t forget to reset all your state variables if the C button is pressed.

6. Tidying Up

Once that is complete, you should have a fully functional app where you can type in an amount, adjust the slider and get all the data you need for paying a tip (if you weren’t living in Australia where it’s unnecessary). As a last step, you should tidy up the UI. The chances are you have cramped text, long thin buttons and so on. Format it so that the buttons are bigger and squarer, the text is larger and just generally make it look more like a calculator.

7. Bill Splitting

Improve the app so it supports splitting the bill. Give it a “number of diners” which can be adjusted with a stepper control and a “Cost per diner:” which reports how much each person must pay.

8. Submission

Show the lecturer your working code before you submit. Submit via Blackboard under “Assessment” in the sidebar.

Solutions

Below are solutions for each section. It's important you try to complete everything yourself since solutions will not be available for future exercises. These are intended to help you while you are still getting to grips with how Xamarin, XAML and C# work but if you copy them verbatim, then you are unlikely to learn and will run into problems over the coming weeks.

Also note that these are not the only solutions. It's perfectly okay if you've done things differently (assuming it works, is readable and so on).

2. Percentage Slider

XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:TipCalculator"
             x:Class="TipCalculator.MainPage">

<StackLayout Orientation="Horizontal">
      <Slider WidthRequest="100"
              Minimum="0"
              Maximum = "100"
              x:Name="percentageSlider"
              ValueChanged="percentageSlider_ValueChanged" />
      <Label Text="0%" x:Name="percentageLabel" />
</StackLayout>

</ContentPage>
```

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace TipCalculator
{
    public partial class MainPage : ContentPage
    {
      public MainPage()
      {
            InitializeComponent();
            percentageSlider.Value = 10;
      }
      private void percentageSlider_ValueChanged(object sender, ValueChangedEventArgs e)
      {
            percentageLabel.Text = String.Format("{0}%", e.NewValue);
      }
```

3. Tip Details

XAML

```
    <StackLayout Orientation="Vertical">
        <Grid>
            <Label WidthRequest="100" Text="Bill:" Grid.Row="0" Grid.Column="0" />
            <Label WidthRequest="100" Text="0" Grid.Row="0" Grid.Column="1"
x:Name="billAmountLabel" />

            <Label WidthRequest="100" Text="Tip:" Grid.Row="1" Grid.Column="0" />
            <Label WidthRequest="100" Text="0" Grid.Row="1" Grid.Column="1"
x:Name="tipAmountLabel" />

            <Label WidthRequest="100" Text="Total:" Grid.Row="2" Grid.Column="0" />
            <Label WidthRequest="100" Text="0" Grid.Row="2" Grid.Column="1"
x:Name="totalAmountLabel" />
        </Grid>

        <StackLayout Orientation="Horizontal">
            <Slider WidthRequest="100" x:Name="percentageSlider"
ValueChanged="percentageSlider_ValueChanged" />
            <Label Text="0%" x:Name="percentageLabel" />
        </StackLayout>
    </StackLayout>
```

C#

```
        float billAmount = 52.80F, tipAmount, totalAmount;

        public MainPage()
        {
            InitializeComponent();
            percentageSlider.Value = 10;
        }

        private void percentageSlider_ValueChanged(object sender, ValueChangedEventArgs
e)
        {
            updateUI();
        }

        private void updateUI ()
        {
            //-- Calculate the bill

            tipAmount = billAmount * (float)percentageSlider.Value / 100.0;
            totalAmount = billAmount + tipAmount;

            //-- Display the tip

            billAmountLabel.Text = String.Format("${0:F2}", billAmount);
            tipAmountLabel.Text = String.Format("${0:F2}", tipAmount);
            totalAmountLabel.Text = String.Format("${0:F2}", totalAmount);
            percentageLabel.Text = String.Format("{0}%", percentageSlider.Value);
        }
```

4. Calculator Buttons

XAML

```
<Grid>
      <Button Text="7" Grid.Row="0" Grid.Column="0" Clicked="CalculatorButtonTapped"/>
      <Button Text="8" Grid.Row="0" Grid.Column="1" Clicked="CalculatorButtonTapped" />
      <Button Text="9" Grid.Row="0" Grid.Column="2" Clicked="CalculatorButtonTapped" />

      <Button Text="4" Grid.Row="1" Grid.Column="0" Clicked="CalculatorButtonTapped" />
      <Button Text="5" Grid.Row="1" Grid.Column="1" Clicked="CalculatorButtonTapped" />
      <Button Text="6" Grid.Row="1" Grid.Column="2" Clicked="CalculatorButtonTapped" />

      <Button Text="1" Grid.Row="2" Grid.Column="0" Clicked="CalculatorButtonTapped" />
      <Button Text="2" Grid.Row="2" Grid.Column="1" Clicked="CalculatorButtonTapped" />
      <Button Text="3" Grid.Row="2" Grid.Column="2" Clicked="CalculatorButtonTapped" />

      <Button Text="0" Grid.Row="3" Grid.Column="0" Clicked="CalculatorButtonTapped" />
      <Button Text="." Grid.Row="3" Grid.Column="1" Clicked="CalculatorButtonTapped" />
      <Button Text="C" Grid.Row="3" Grid.Column="2" Clicked="CalculatorButtonTapped" />
</Grid>
```

C#

```
private void CalculatorButtonTapped(object sender, EventArgs e)
{
      Button button = (Button)sender;
      Console.WriteLine(button.Text);
}
```

5. Entering a Bill Amount

C#

```
bool hasStarted = false, hasTypedDecimal = false;
int numberOfDecimalDigits = 0
```

```
private void reset () {
      hasStarted = false;
      hasTypedDecimal = false;
      numberOfDecimalDigits = false;
      billAmount = 0.0F;
      tipAmount = 0.0F;
      totalAmount = 0.0F;

      updateUI();
}

private void CalculatorButtonTapped(object sender, EventArgs e)
{
      Button button = (Button)sender;

      //-- Reset everything if “C” is tapped

      if (button.Text == “C”)
      {
            reset();
            return;
      }

      //-- Check if “0” is a valid keypress at this point

      if (button.Text == “0” && hasStarted == false)
      {
            return;
      }

      //-- If a decimal has been typed, check if it’s valid

      if (button.Text = “.” && hasTypedDecimal)
      {
            return;
      }

      //-- If we’ve made it this far, we have officially started getting a valid bill

      if (!hasStarted)
      {
            hasStarted = true;
      }

      //-- Handle decimal...

      if (button.Text == “.”)
      {
            hasTypedDecimal == true;
        }

        //-- Handle numbers…

        else  // anything else should be a number
        {
            if (hasTypedDecimal) {
                if (numberOfDecimalDigits < 2)
                {
                    numberOfDecimalDigits++;
                }
            }
        }

        //— Update the bill amount.

        if (hasTypedDecimal)
        {
           //— If the user is typing a decimal, calculate the amount to add

           float multiplier = numberOfDecimalDigits == 2 ? 0.01F : 0.1F;
           billAmount += multiplier * Int32.Parse(button.Text);
        }

        else {

           //-- If the user is typing a whole number, then shift the current
           //-- bill amount to make room for the new number

           billAmount *= 10;

           //-- Convert the button label to an integer and add it

           billAmount += Int32.Parse(button.Text);
        }

        updateUI();
}
```