



Android, iOS and Hybrid Applications

Mobile-App Development

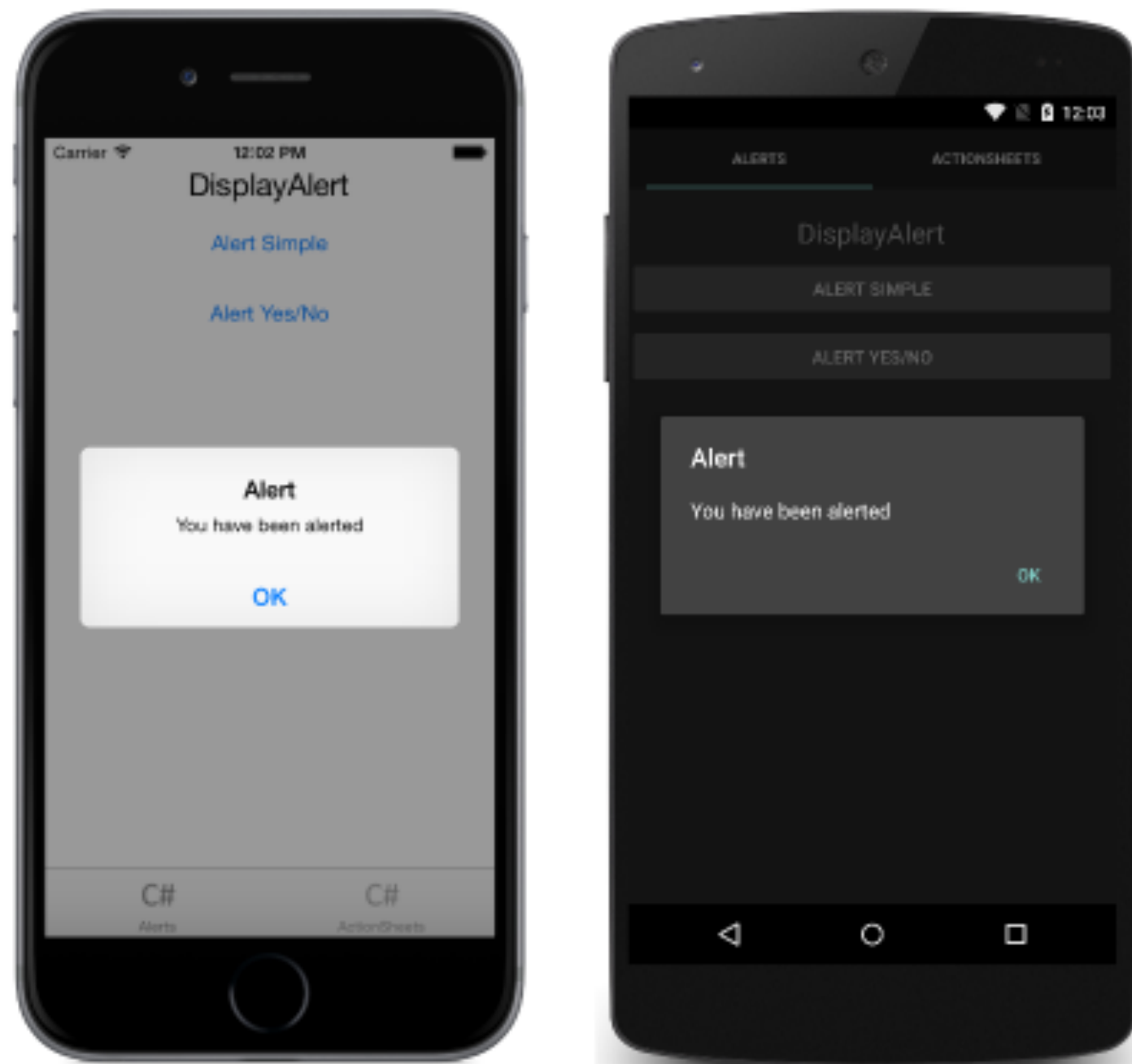
DAY 3

- ▶ Dialogs
- ▶ Styling
- ▶ Inversion of Control (IOC)
- ▶ Testing

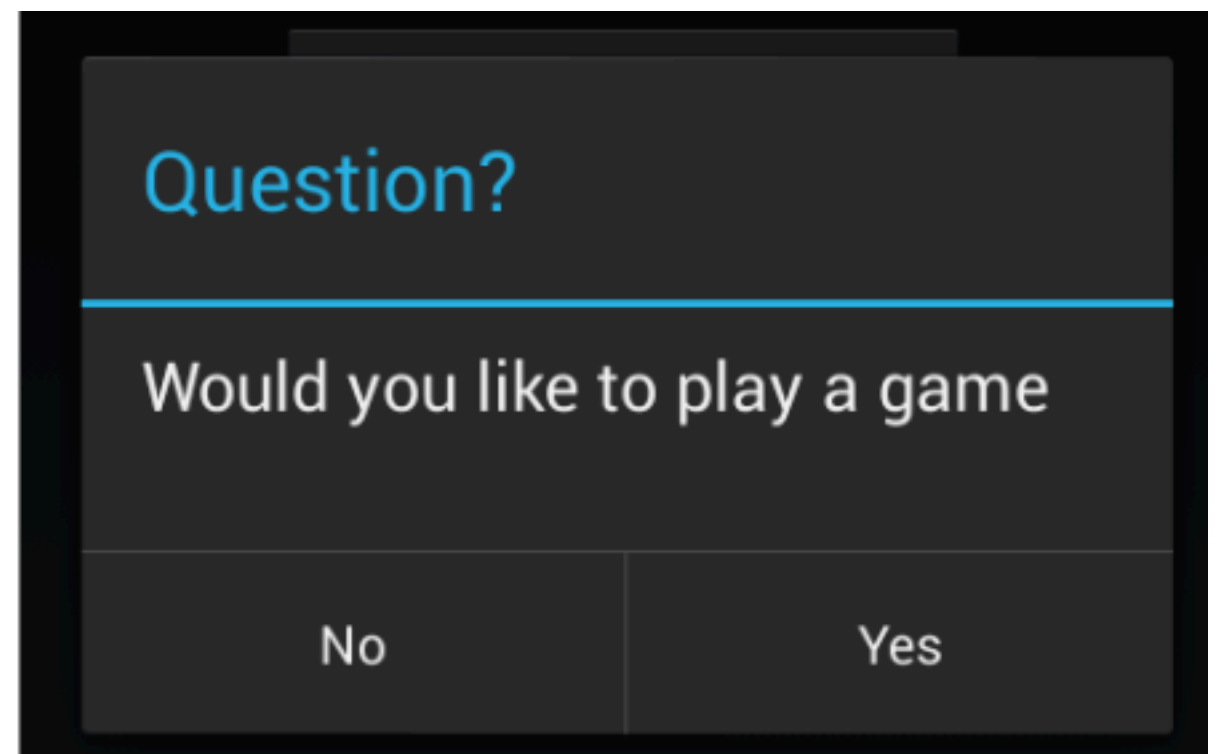
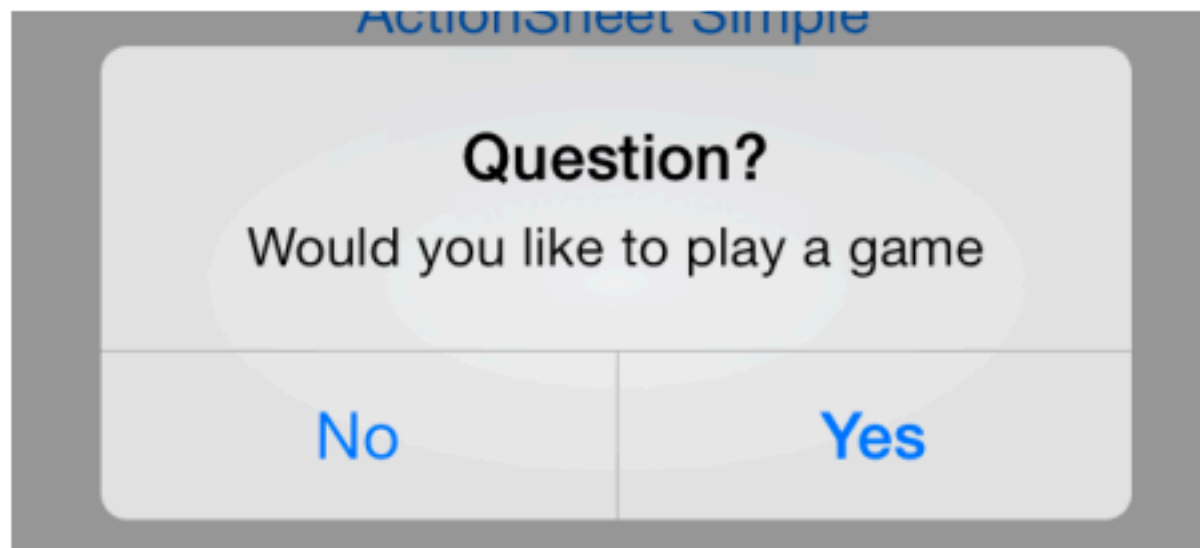
DIALOGS (POP-UPS)

- ▶ Call `DisplayAlert("", "")` on any Page
 - ▶ Ask "questions" with the overloads
 - ▶ *await* the result
-
- ▶ Use *Action Sheets* for behavior like a "DropDown"

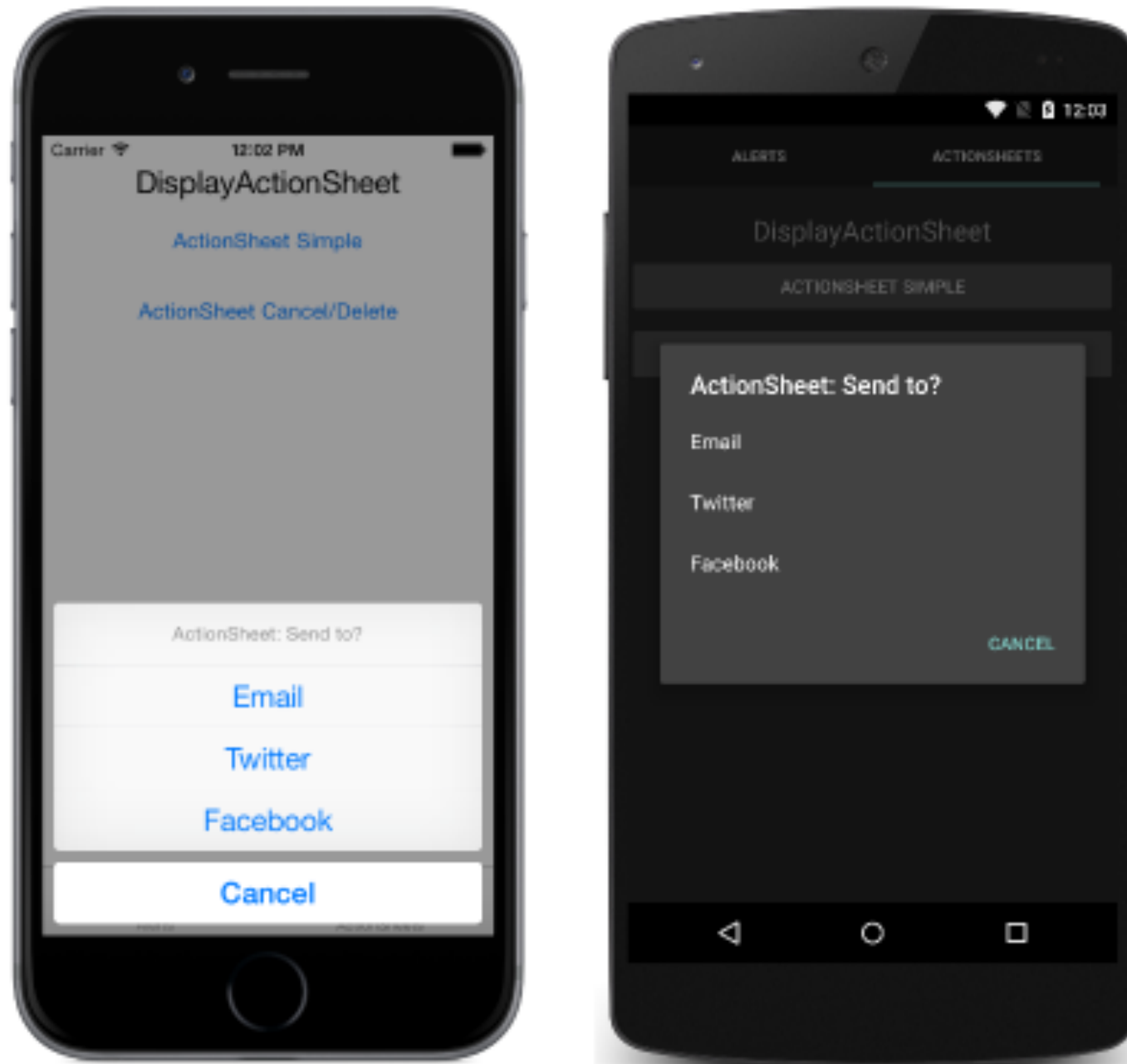
DIALOGS



DIALOGS



DIALOGS



DIALOGS – CODE SAMPLES

```
DeleteItemCommmand = new Command<Page>(async page =>
{
    var result = await page.DisplayAlert("Confirm", "Are you sure?", "YES", "NO");
    if (result)
    {
        Items.Remove(SelectedItem);
    }
});
```

```
async void OnActionSheetSimpleClicked(object sender, EventArgs e)
{
    string action = await DisplayActionSheet("ActionSheet: Send to?", "Cancel",
                                             null, "Email", "Twitter", "Facebook");
    Debug.WriteLine("Action: " + action);
}
```

DIALOGS – CODE SAMPLES

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="Todo.Views.TodoListPage"
  xmlns:local="clr-namespace:Todo.Converters;assembly:Todo"
  xmlns:viewModels="clr-namespace:Todo.ViewModels;assembly:Todo"
  x:DataType="viewModels:TodoListViewModel"
  x:Name="Page">
  <ContentPage.Content>

    <Button Text="Delete" WidthRequest="100" Margin="20,0,0,0"
      Command="{Binding DeleteItemCommand}" CommandParameter="{x:Reference Page}" />

  </ContentPage.Content>
</ContentPage>
```


DIALOGS – API

```
DisplayAlert(string title, string message, string cancel);
```

```
DisplayAlert(string title, string message,  
             string accept, string cancel);
```

```
DisplayActionSheet(string title, string cancel,  
                  string destruction, params string[] buttons);
```

MOBILE-DEVELOPMENT

QUESTIONS?

PRACTICE

- ▶ Example
- ▶ Use a Dialog in your solution
- ▶ Pass the Page via the command parameter

STYLING

- ▶ You can use XAML or CSS
- ▶ We're going to focus on XAML
- ▶ [Check online](#) for which standard properties are supported by the various controls

STYLING – HIERARCHY

- ▶ Directly on an Element (akin to CSS inline styles)
- ▶ *Explicit Styles*: set the “Style” directly on an element
- ▶ *Implicit Styles*: a default style applied via the *TargetType*

STYLING ON THE ELEMENT – EXAMPLE

```
<Label  
    Grid.Column="2"  
    Text="X"  
    TextColor="Red" />
```


STYLING EXPLICIT – EXAMPLE

```
<ContentPage
  xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="Todo.Views.TodoListPage"
  Title="List">
  <ContentPage.Style>
    <Style>
      <Setter
        Property="BackgroundColor"
        Value="Black" />
    </Style>
  </ContentPage.Style>
```

STYLING EXPLICIT – EXAMPLE

App.xaml

```
<Application
  xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="Todo.App">
  <Application.Resources>
    <ResourceDictionary>
      <Color x:Key="Background">
        Black
      </Color>
      <Style x:Key="ContentPageStyle" TargetType="ContentPage">
        <Setter Property="BackgroundColor" Value="Black" />
      </Style>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

SomePage.xaml

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  x:Class="Todo.Views.TodoListPage"
  x:Name="Page"
  Title="List"
  Style="{StaticResource ContentPageStyle}">
```

STYLING IMPLICIT – EXAMPLE

```
<Style
  TargetType="Button">
  <Setter
    Property="BackgroundColor"
    Value="#3541a0" />
  <Setter
    Property="TextColor"
    Value="White" />
  <Setter
    Property="HeightRequest"
    Value="50" />
</Style>
```

STYLING – POSSIBLE VALUES

- ▶ Button
 - ▶ BackgroundColor
 - ▶ BorderRadius
 - ▶ BorderWidth
 - ▶ BorderColor
 - ▶ TextColor

STYLING – POSSIBLE VALUES

- Entry
 - TextColor
 - FontSize
 - FontFamily
 - PlaceholderColor

STYLING – POSSIBLE VALUES

- Picker
 - TextColor
 - FontSize
 - FontFamily
 - TitleColor

STYLING – POSSIBLE VALUES

- Label
 - TextColor
 - BackgroundColor
 - FontSize
 - FontFamily
 - TextDecorations

STYLING – ADDITIONAL RESOURCES

- [Triggers](#)
- [Custom Renderer](#)
- [Effects](#)

MOBILE-DEVELOPMENT

QUESTIONS?

PRACTICE

- ▶ Example
- ▶ Style your App

IOC

- ▶ We would like to work with *abstractions*
- ▶ IOC: [I]nversion [O]f [C]ontrol is such a *pattern*
- ▶ A *container* has *registrations* from
 - ▶ An *abstraction*...
 - ▶ ...to an implementation
- ▶ A container *resolves* requests for *abstractions* by providing *implementations*
- ▶ See [Dependency Injection](#)
- ▶ See [How do I DI?](#) presentation (2018)

The background of the slide is a dark gray with a complex, white, and light gray circuit board pattern. The pattern consists of various lines, nodes, and rectangular shapes, resembling a technical drawing of a PCB. The lines are of different thicknesses and connect various points, creating a network-like structure. Some shapes are solid white, while others are light gray, adding depth to the design.

How do I DI?

IOC – DIALOGSERVICE – INTERFACE

```
public interface IDialogService
{
    Task Show(string title, string message);

    Task<bool> Show(string title, string message, string positive, string negative);
}
```

IOC – DialogService – IMPLEMENTATION

```
public class DialogService : IDialogService
{
    public DialogService(Page page)
    {
        _page = page;
    }
}
```

```
public async Task Show(string title, string message)
{
    await _page.DisplayAlert(title, message, "Cancel");
}
```

```
public async Task<bool> Show(string title, string message, string positive, string negative)
{
    return await _page.DisplayAlert(title, message, positive, negative);
}
```

```
private readonly Page _page;
}
```

IOC – DIALOGSERVICE – TESTIMPLEMENTATION

```
public class MockDialogService : IDialogService
{
    public Task Show(string title, string message)
    {
        return Task.CompletedTask;
    }
}
```

```
public Task<bool> Show(string title, string message, string positive, string negative)
{
    return Task.FromResult(true);
}
}
```

IOC – DIALOGSERVICE – USAGE

```
DeleteItemCommand = new Command(async () =>
{
    if (await dialogService.Show("Confirm", "Are you sure you want to delete the item?", "Yes", "No"))
    {
        Items.Remove(Items.First(x => x.IsSelected));
    }
});
```

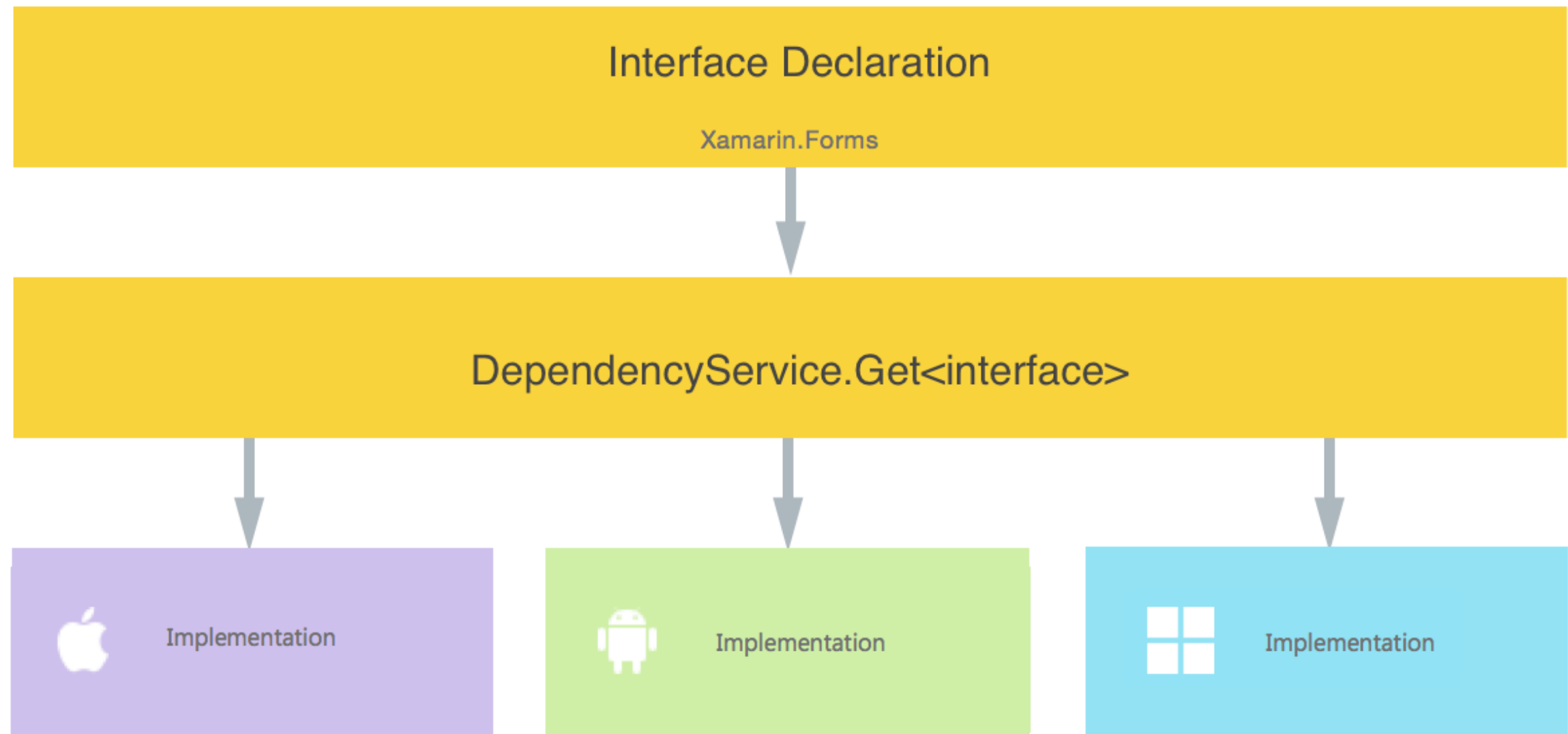
IOC – WORKFLOW

- ▶ Register your Services
 - ▶ `Container.Register<Interface, Implementation>()`
- ▶ Seal the container - no more registrations after this point
- ▶ Resolve services using the container
 - ▶ `Container.Resolve<Interface>()`

IOC – SUMMARY OF ADVANTAGES

- ▶ Replace your services for testing
- ▶ No more “new” throughout the code
- ▶ Use different implementations based on a condition
 - ▶ iOS vs. Android
 - ▶ Production vs. Testing
- ▶ Constructor injection
 - ▶ is easy to understand
 - ▶ make it easy to identify dependencies

IOC - XAMARIN FORMS



XAMARIN FORMS – IOC

Shared:

```
public interface ISomeService
{
    void Foo();
}
```

Android:

```
using System;
using FormsTesting.Droid;
using Xamarin.Forms;

// Android specific implementation. Registration via attribute.
[assembly: Dependency(typeof(SomeService))]
namespace FormsTesting.Droid
{
    public class SomeService : ISomeService
    {
        public void Foo()
        {
            throw new NotImplementedException();
        }
    }
}
```

→ Same for iOS

IOC – XAMARIN FORMS

- ▶ We *can* but we don't *have* to use it
- ▶ It's a pretty simple container with a lot of limitations
- ▶ You **have** to use it for custom controls

IOC – SIMPLE INJECTOR

- ▶ SimpleInjector is more powerful
- ▶ It's cross-platform
- ▶ It has excellent documentation
- ▶ We use it in the sample project

IOC – SIMPLE INJECTOR

```
// Register services we need to set up our application.
Services.RegisterInstance<Page>(navigationPage);
Services.RegisterSingleton<IViewMapper, ViewMapper>();
Services.RegisterSingleton<ITodoViewModelFactory, TodoViewModelFactory>();
Services.RegisterSingleton<ITodoItemProvider, TodoItemProvider>();
Services.RegisterSingleton<MainViewModel>();
Services.RegisterSingleton<TodoListViewModel>();
Services.Register<TodoItemViewModel>();
```

```
Services.GetInstance<TodoListViewModel>()
```

```
public TodoListViewModel(INavigation navigation, IViewMapper viewMapper,
ITodoViewModelFactory viewModelFactory, ITodoItemProvider provider)
{
    // Constructor
}
```

MOBILE-DEVELOPMENT

QUESTIONS?

TESTING

- ▶ Use a "NUnit Test Project"
- ▶ Reference your shared project
- ▶ One test class per service
- ▶ Feel free to create base classes or helper methods

TESTING – SETUP

```
[TestFixture]
public class Tests
{
    [OneTimeSetUp]
    public void Setup()
    {
        // Potentially register different services to set up a "predictable" test environment.
        App.Services.RegisterInstance(new NavigationPage().Navigation);
        App.Services.Register<IViewMapper, ViewMapper>(Lifestyle.Singleton);
        App.Services.Register<ITodoViewModelFactory, TodoViewModelFactory>(Lifestyle.Singleton);
        App.Services.Register<ITodoItemProvider, TodoItemProvider>(Lifestyle.Singleton);
        App.Services.Register<MainViewModel>(Lifestyle.Singleton);
        App.Services.Register<TodoListViewModel>(Lifestyle.Singleton);
        App.Services.Register<TodoItemViewModel>(Lifestyle.Transient);

        App.Services.Register<IDialogService, MockDialogService>(Lifestyle.Singleton);
    }
}
```


TESTING – TEST

```
[TestFixture]
```

```
public class Tests
```

```
{
```

```
    // Setup excluded
```

```
[Test]
```

```
public void TestEmptyTodoCantBeSaved()
```

```
{
```

```
    var listViewModel = App.Services.GetInstance<TodoListViewModel>();
```

```
    var todoItemViewModel = App.Services.GetInstance<ITodoViewModelFactory>()  
        .Create(new TodoItem(), listViewModel);
```

```
    Assert.That(todoItemViewModel.SaveCommand.CanExecute(null), Is.False);
```

```
    todoItemViewModel.Title = "Title";
```

```
    Assert.That(todoItemViewModel.SaveCommand.CanExecute(null), Is.True);
```

```
}
```

```
}
```

MOBILE-DEVELOPMENT

QUESTIONS?

TESTING & IoC

- ▶ Include an IOC in your app
- ▶ Move your dependencies into the IOC
- ▶ Register different services for your test scenarios
- ▶ Examples:
 - ▶ On/Offline service
 - ▶ Item Provider or similar that connects to an API/DB
 - ▶ Any UI-specific/related services

ADDITIONAL TASKS

- ▶ Apply some of the additional styling options
- ▶ Use CSS to style something
- ▶ Expand the tests with TestCases