Android, iOS and Hybrid Applications

Mobile-Development

Day 3

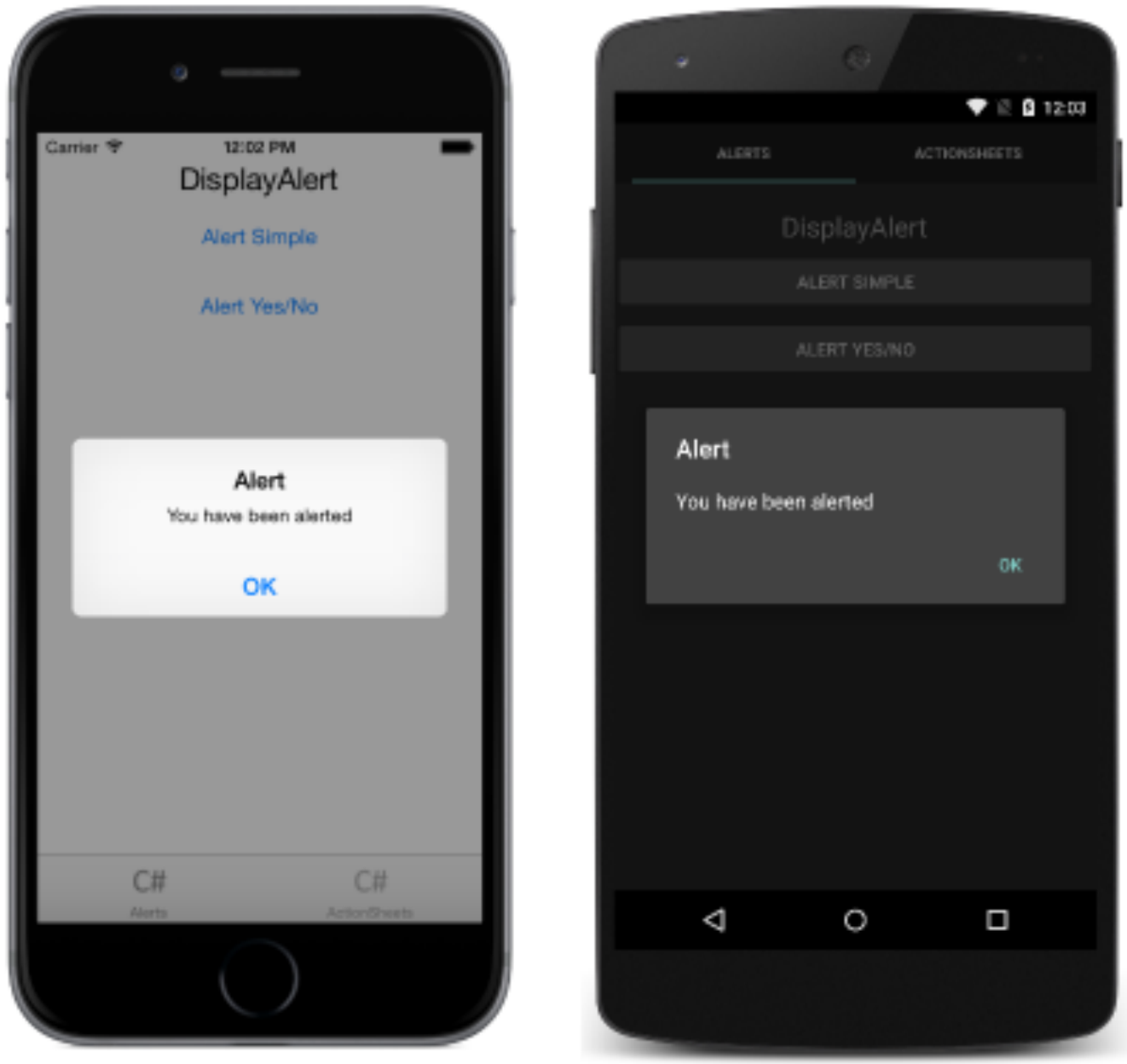
- Dialogs
- Styling
- Inversion of Control (IOC)
- Testing

Dialogs (Pop-Ups)

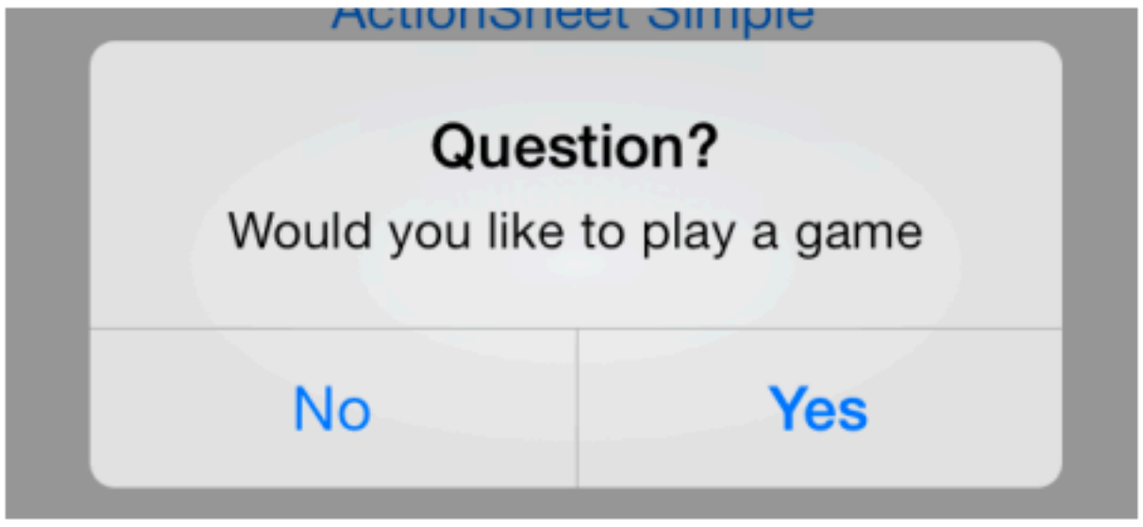
- Call DisplayAlert(“”, “”) on any Page
- Ask questions with the overloads
- “Await” the result
- Action Sheets for a “DropDown” like behaviour

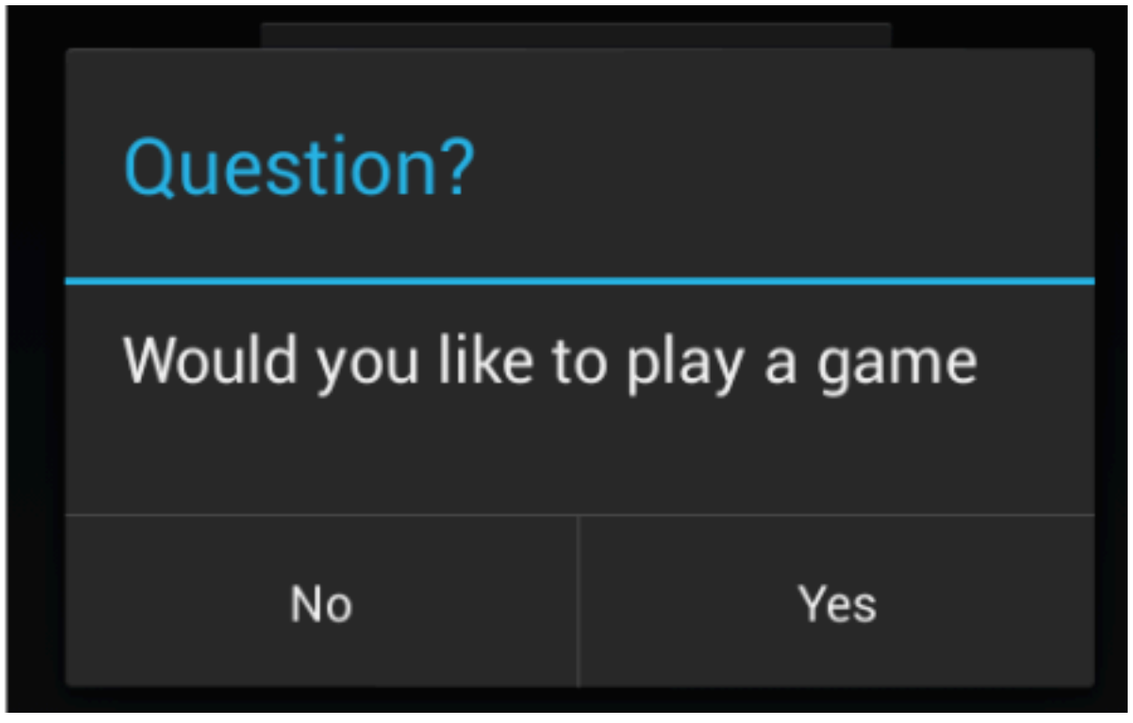
https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/pop-ups

DIALOGS

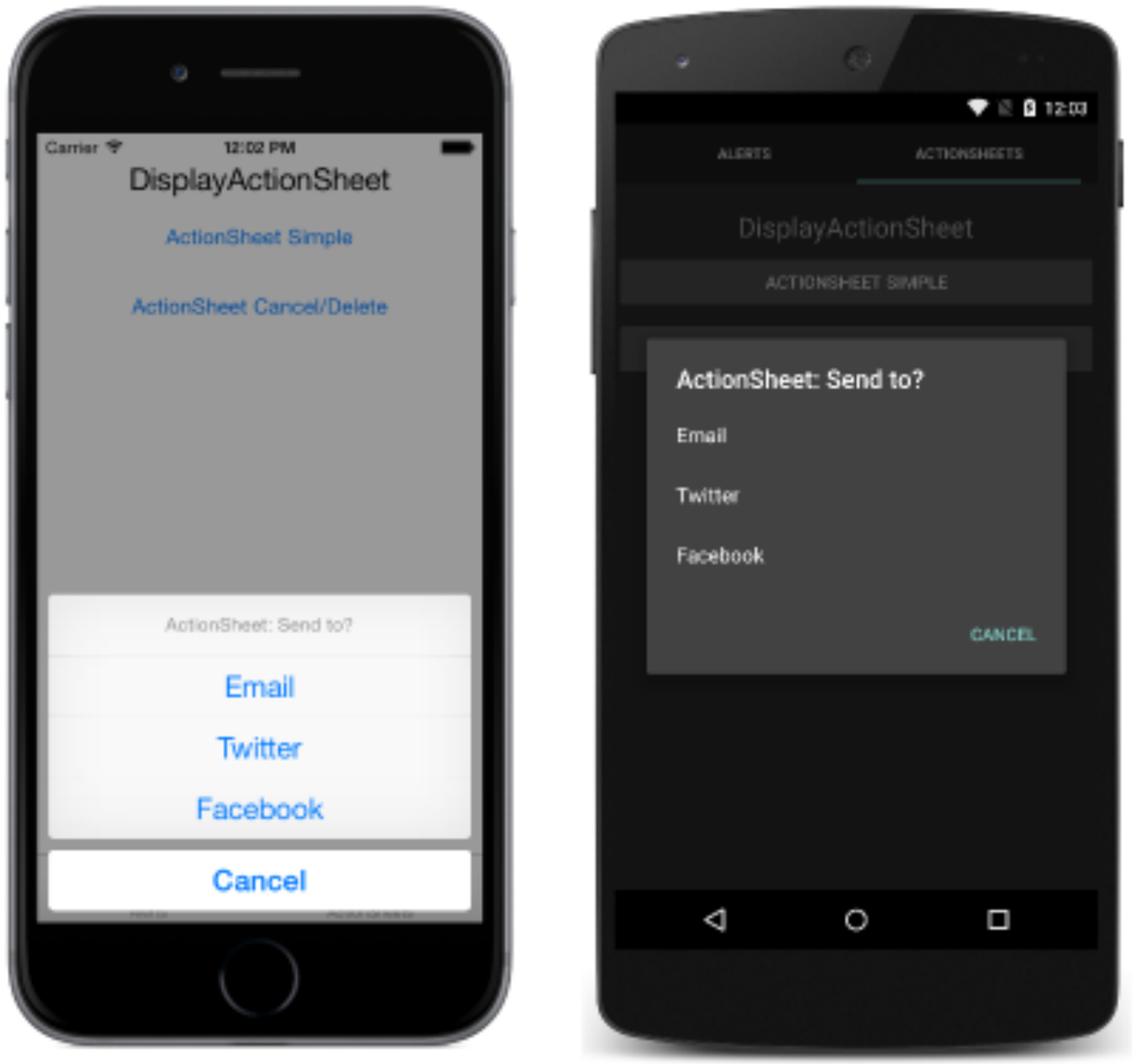

Carrier
12:02 PM
DisplayAlert
Alert Simple
Alert Yes/No
Alert
You have been alerted
OK
C#
Alerts
C#
ActionSheets
12:00
ALERTS
ACTIONSHEETS
DisplayAlert
ALERT SIMPLE
ALERT YES/NO
Alert
You have been alerted
OK

DIALOGS


ActionSheet Simple
Question?
Would you like to play a game
No
Yes


Question?
Would you like to play a game
No
Yes

DIALOGS


Carrier
12:02 PM
DisplayActionSheet
ActionSheet Simple
ActionSheet Cancel/Delete
ActionSheet: Send to?
Email
Twitter
Facebook
Cancel
12:00
ALERTS
ACTIONSHEETS
DisplayActionSheet
ACTIONSHEET SIMPLE
ActionSheet: Send to?
Email
Twitter
Facebook
CANCEL

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 - 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);
```

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 what properties are supported by the various types

https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/styles/xaml/index

STYLING – HIERARCHY

- Directly on an Element
- Explicit Styles - set 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"
    Style="{StaticResource ContentPageStyle}">
```

```
<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 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

https://docs.microsoft.com/en-gb/dotnet/api/xamarin.forms.button?view=xamarin-forms

Styling – Possible Values

- Entry
  - TextColor
  - FontSize
  - FontFamily
  - Placeholder
  - PlaceholderColor

https://docs.microsoft.com/en-gb/dotnet/api/xamarin.forms.entry?view=xamarin-forms

STYLING – POSSIBLE VALUES

- Picker
  - TextColor
  - FontSize
  - FontFamily
  - Title
  - TitleColor

https://docs.microsoft.com/en-gb/dotnet/api/xamarin.forms.picker?view=xamarin-forms

Styling – Possible Values

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

https://docs.microsoft.com/en-gb/dotnet/api/xamarin.forms.label?view=xamarin-forms

Practice

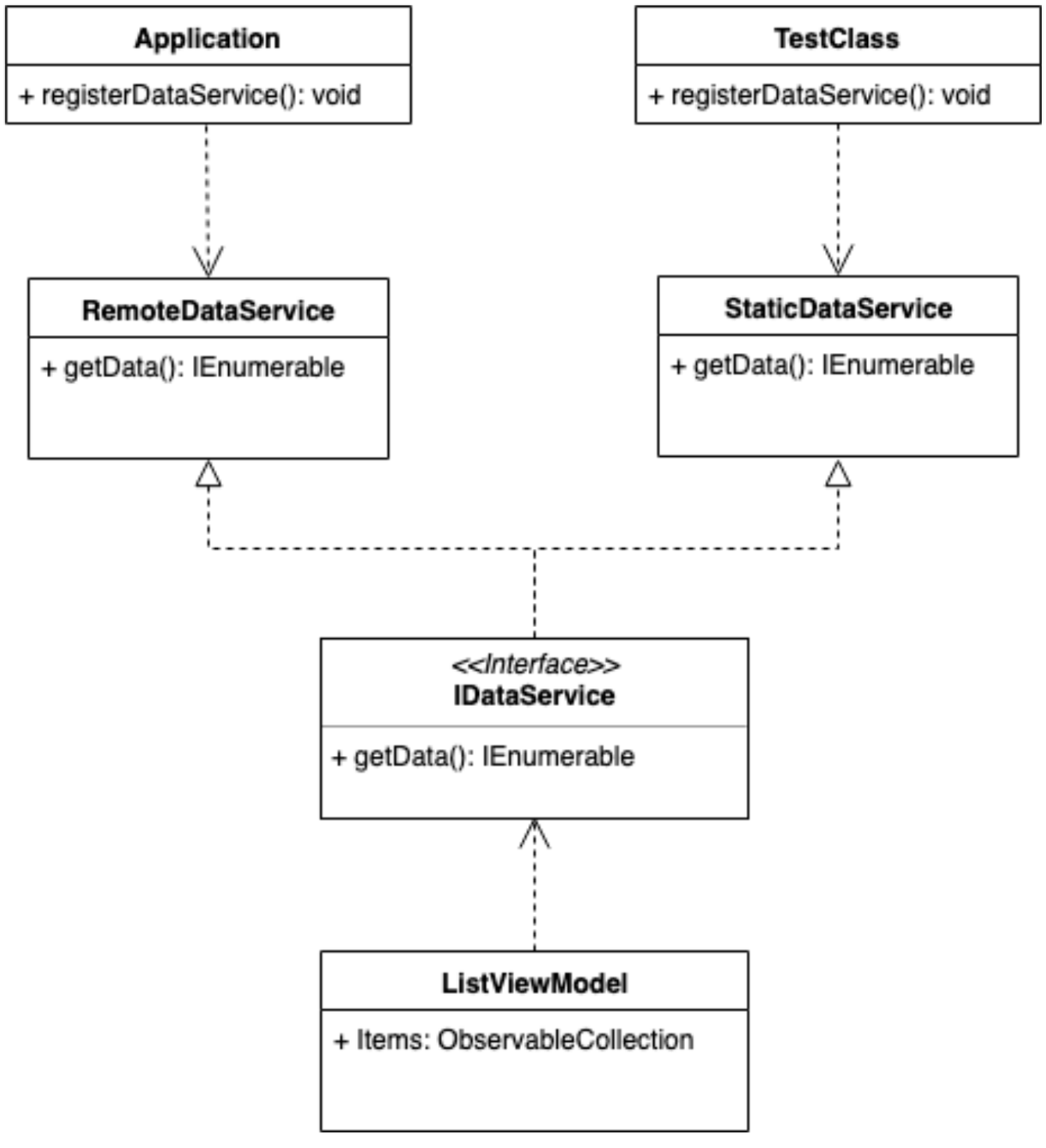
- Example
- Apply some basic styles to your App

IOC

- Inversion of Control
- Dependency Injection as a specialised version
- We don’t want to work with concrete implementations
- A container holds the registrations and resolves them

https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/dependency-injection

IOC


Application
+ registerDataService(): void
TestClass
+ registerDataService(): void
RemoteDataService
+ getData(): IEnumerable
StaticDataService
+ getData(): IEnumerable
<<Interface>>
IDataService
+ getData(): IEnumerable
ListViewModel
+ Items: ObservableCollection

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>()

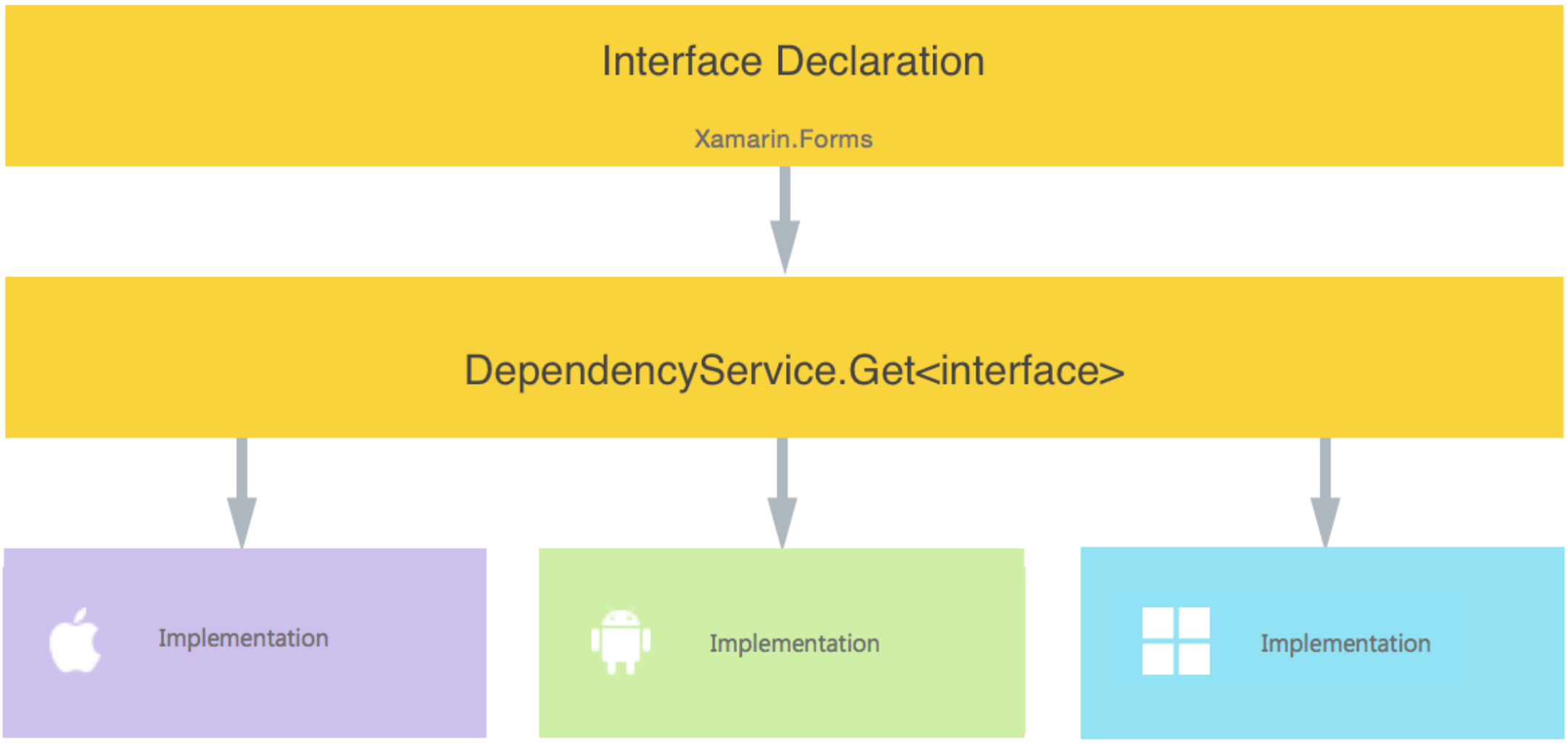
IOC – TRANSIENT VS SINGLETON

- Singleton exists only once per app/container
- Transient objects are created with every request
- Lifestyle mismatch! Singleton which depends on transient!

IOC – Advantages

- Replace your services for testing
- No more “new” all through the code
- Use different implementations based on a condition (iOS, Android for example)
- Constructor injection is easy to understand and see 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();
        }
    }
}
```

-> Do the same for iOS

IOC – Xamarin Forms

- We can but we don’t need 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

- There’re a lot of IoC frameworks out there
- Cross Platform
- Good documentation
- Used in the sample project

https://simpleinjector.readthedocs.io/en/latest/quickstart.html

IOC – Simple Injector

```
// Register services we need to setup our application.
Services.RegisterInstance(navigationPage.Navigation);
Services.Register<IViewMapper, ViewMapper>(Lifestyle.Singleton);
Services.Register<ITodoViewModelFactory, TodoViewModelFactory>(Lifestyle.Singleton);
Services.Register<ITodoItemProvider, TodoItemProvider>(Lifestyle.Singleton);
Services.Register<MainViewModel>(Lifestyle.Singleton);
Services.Register<TodoListViewModel>(Lifestyle.Singleton);
Services.Register<TodoItemViewModel>(Lifestyle.Transient);
```

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

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

Testing

- Use a standard .NET Core Unit Test project
- Reference your shared project
- One test class per service
- Feel free to create base classes or helper methods

https://docs.microsoft.com/en-us/dotnet/core/testing/

TESTING & IOC

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