



AND205

Navigation Patterns

- ▶ Lecture will begin shortly
- ▶ Download class materials from university.xamarin.com

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

Xamarin may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any license agreement from Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.

© 2016 Xamarin. All rights reserved.

Xamarin, MonoTouch, MonoDroid, Xamarin.iOS, Xamarin.Android, and Xamarin Studio are either registered trademarks or trademarks of Xamarin in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

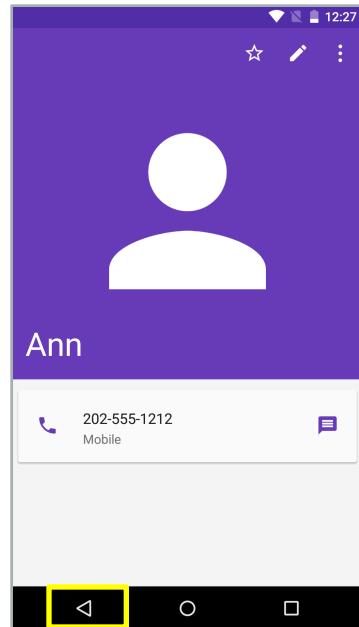
Objectives

1. Implement Stack navigation
2. Introduce Fragments
3. Implement Tab navigation
4. Introduce **ActionBar**
5. Implement Drawer navigation

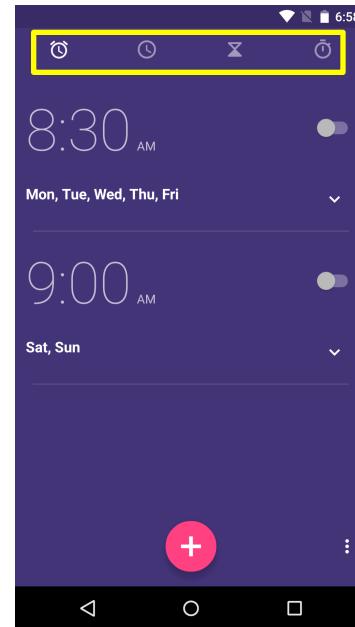


Navigation Patterns

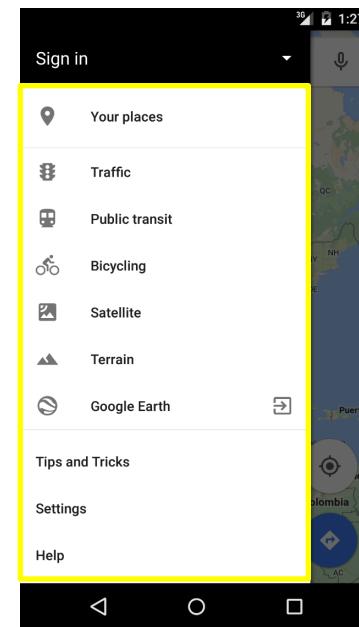
- ❖ Android apps use several common navigation patterns



Stack



Tab



Drawer



Implement Stack Navigation



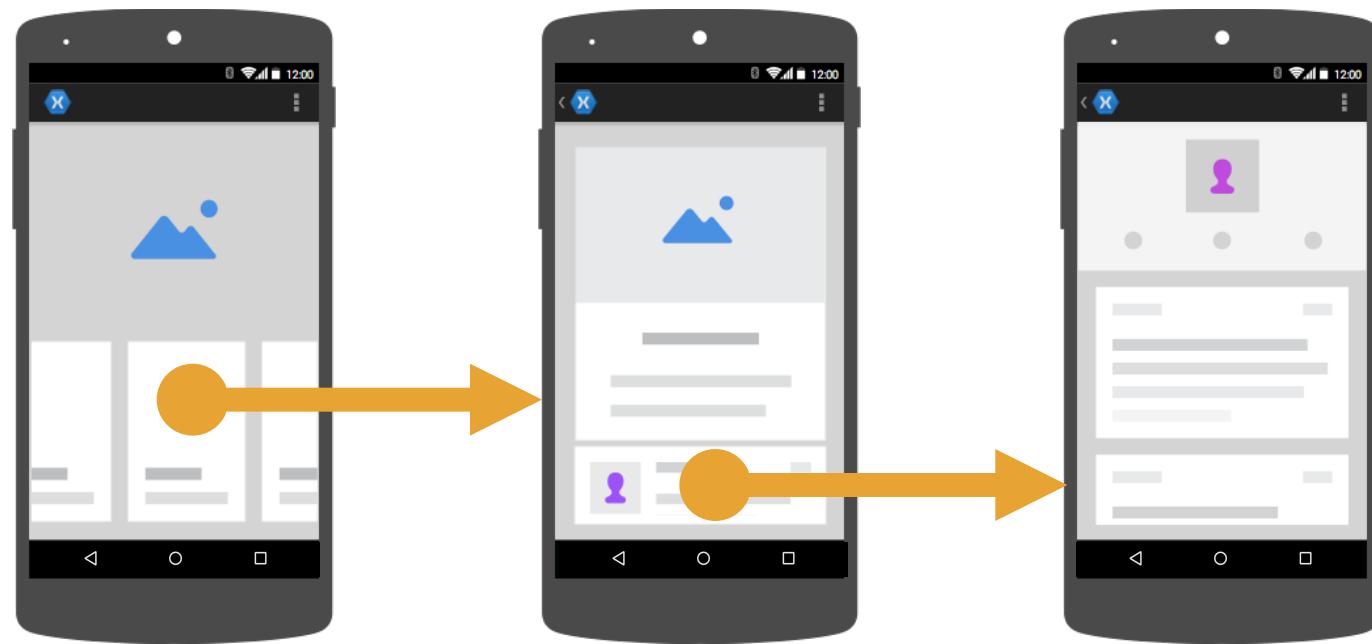
Tasks

1. Implement Forward navigation
2. Use the back-button for Back navigation



Forward Navigation [definition]

- ❖ *Forward navigation* is the process of moving from one activity to another



Evolve 2014

Forward Navigation [implementation]

- ❖ Use an Intent to navigate forward

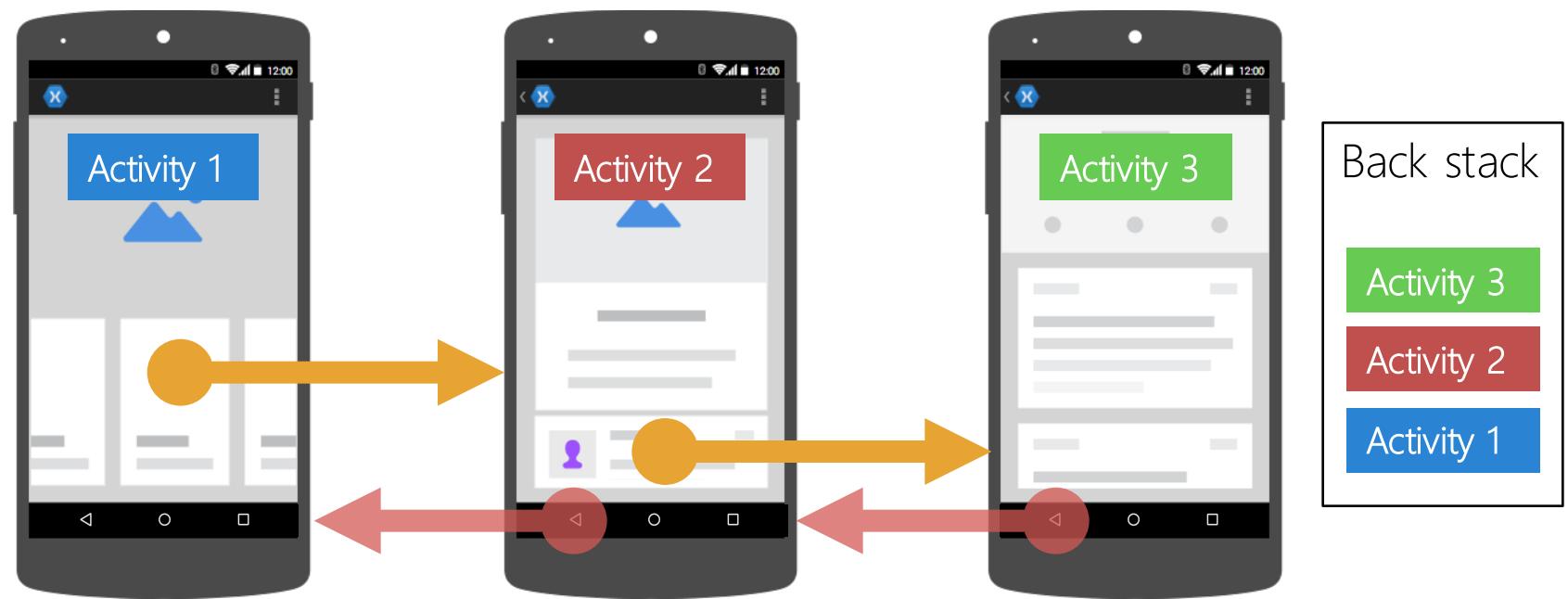
```
var intent = new Intent(this, typeof(SessionActivity));  
  
intent.PutExtra("Title", title);  
  
StartActivity(intent);
```



Launches the new Activity and adds
the previous one to the back stack

What is the back-stack?

- ❖ The *back-stack* is a historical record of the user's live Activities



Back Navigation

- ❖ *Back navigation* moves through the screens in the back-stack
 - Screen progression tracked by default, so no additional work to implement



All Android devices provide either hardware or software back button



Individual Exercise

Stack Navigation



Summary

1. Implement Forward navigation
2. Use the back-button for Back navigation





Introduce Fragments



Tasks

1. Add a **Fragment** to a **FrameLayout** dynamically

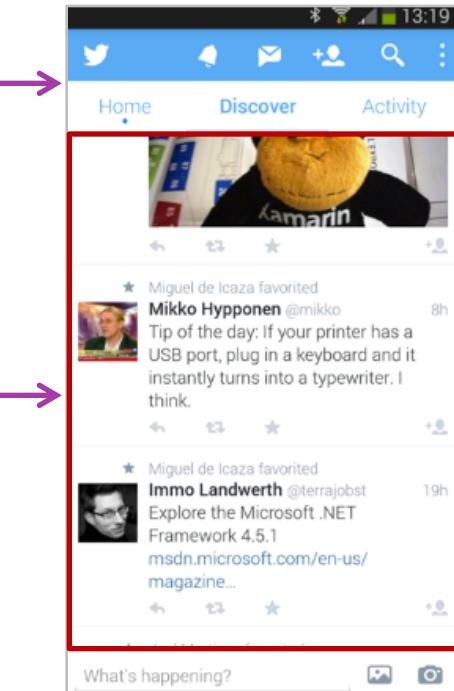


Motivation

- ❖ Activities are too large to be the core building blocks of a dynamic UI

Want to keep this
part the same...

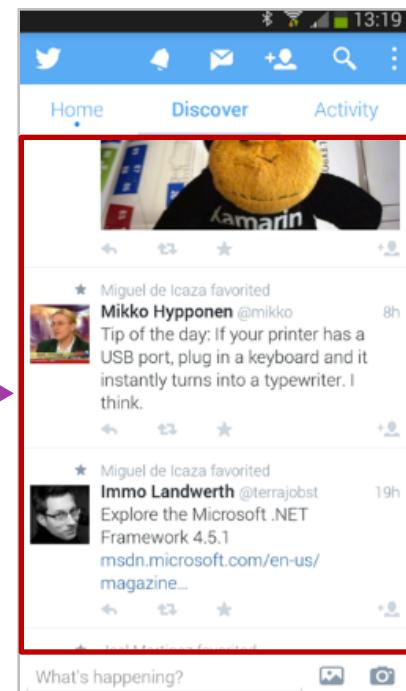
...while swapping
new content and
behavior in here



What is **FrameLayout**? [concept]

- ❖ A **FrameLayout** is a container that is intended to hold a single child, it is common to set the child from code

Use a **FrameLayout**
here and replace its
child-view dynamically



What is FrameLayout? [code]

- ❖ **FrameLayout** methods let you update its child view

No children
In AXML

```
<FrameLayout android:id = "@+id/myFrame" ... />
```

```
void ShowInFrame(string message)
{
    var frame = FindViewById<FrameLayout>(Resource.Id.myFrame);

    if (frame.ChildCount > 0)
        frame.RemoveViewAt(0);

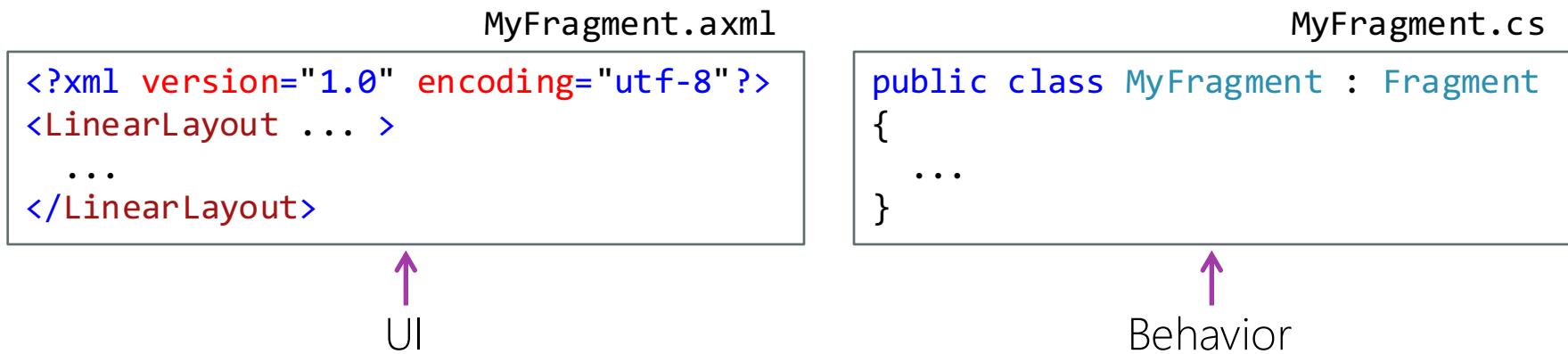
    var tv = new TextView(this) { Text = message };
    frame.AddView(tv);
}
```

Remove old child →

Add new child →

What is a Fragment?

- ❖ A *Fragment* is a unit of UI + behavior intended for use with dynamic UI



Fragments and Activities

- ❖ Fragments are hosted inside Activities, the **Activity** class has built-in support for this

```
public class Activity : ...
{
    ...
    public virtual FragmentManager FragmentManager { get; }
}
```



Helps you dynamically add/remove
fragments from your Activity's UI

Fragment transactions

- ❖ Android requires that dynamic changes to an Activity's fragments be done inside a transaction

```
public abstract class FragmentManager
{
    ...
    public abstract FragmentTransaction BeginTransaction();
}
```

Update
your UI's →
fragments

```
public abstract class FragmentTransaction
{
    ...
    public abstract FragmentTransaction Remove (Fragment fragment);
    public abstract FragmentTransaction Add     (int containerViewId, Fragment fragment);
    public abstract FragmentTransaction Replace(int containerViewId, Fragment fragment);

    public abstract int Commit();
}
```

The Id of the **FrameLayout** that will hold your fragment

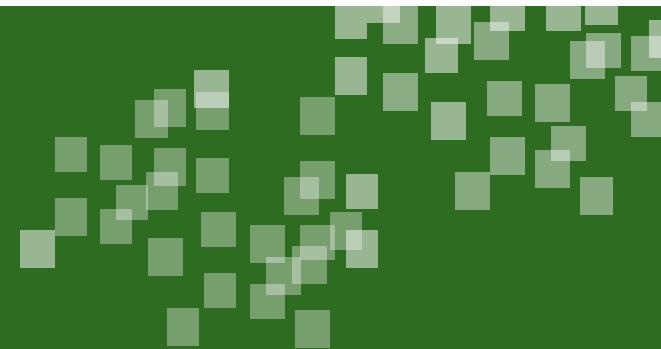
How to replace a fragment

- ❖ **FragmentTransaction** handles the details of loading a new fragment into your UI

```
public class MainActivity : Activity
{
    ...
    void ShowFragment()
    {
        var fragment = new MyFragment();

        var transaction = base.FragmentManager.BeginTransaction();
        transaction.Replace(Resource.Id.myFrame, fragment);
        transaction.Commit();
    }
}
```

Add to UI →



Flash Quiz



Flash Quiz

- ① Which term describes the common use of **FrameLayout**?
- a) Placeholder
 - b) Root
 - c) Border
-

Flash Quiz

- ① Which term describes the common use of **FrameLayout**?
- a) Placeholder
 - b) Root
 - c) Border
-

Flash Quiz

- ② A **Fragment** is bigger than an **Activity**.
- a) True
 - b) False
-

Flash Quiz

- ② A **Fragment** is bigger than an **Activity**.
- a) True
 - b) False
-

Summary

1. Add a **Fragment** to a **FrameLayout** dynamically



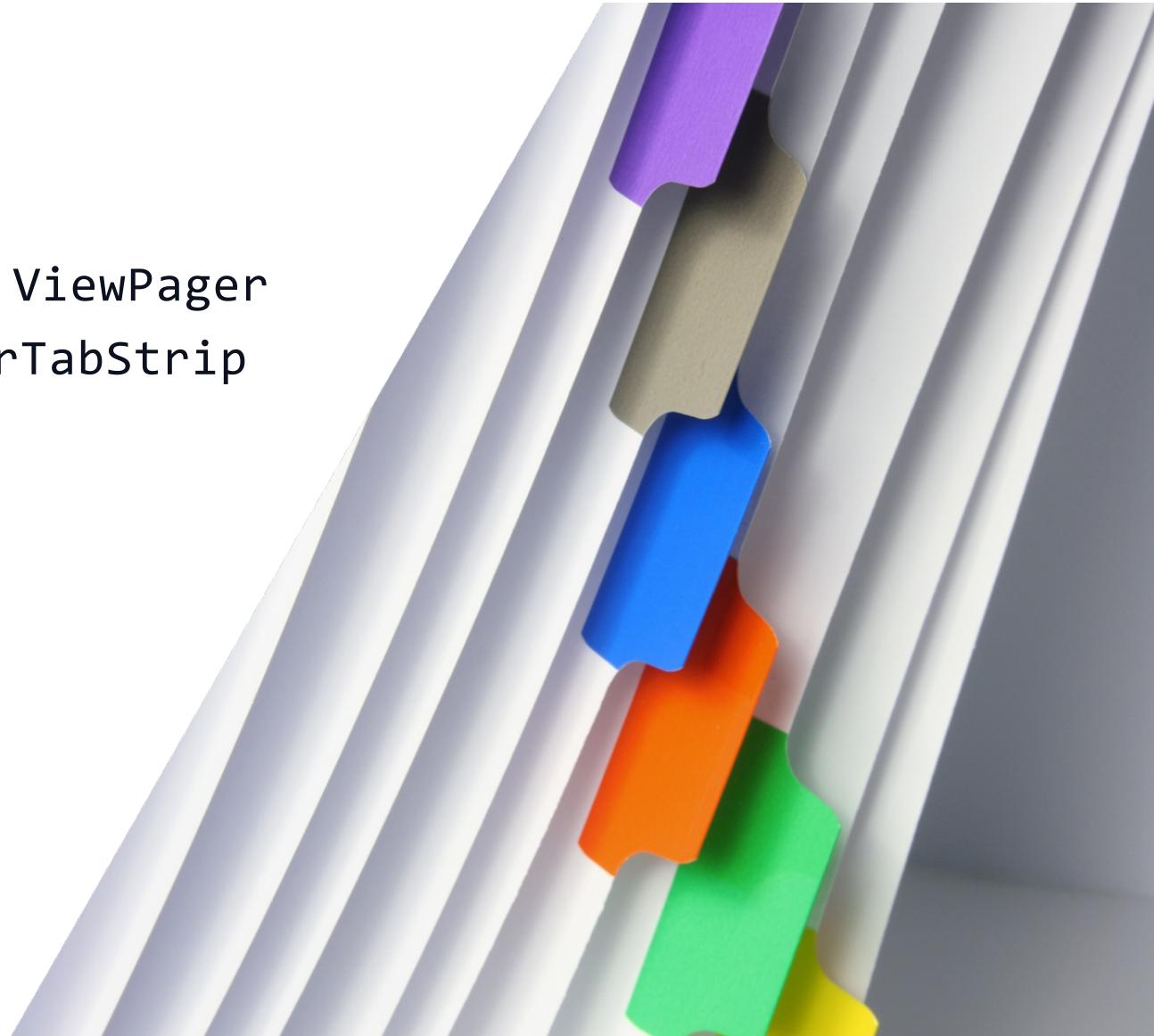


Implement Tab Navigation



Tasks

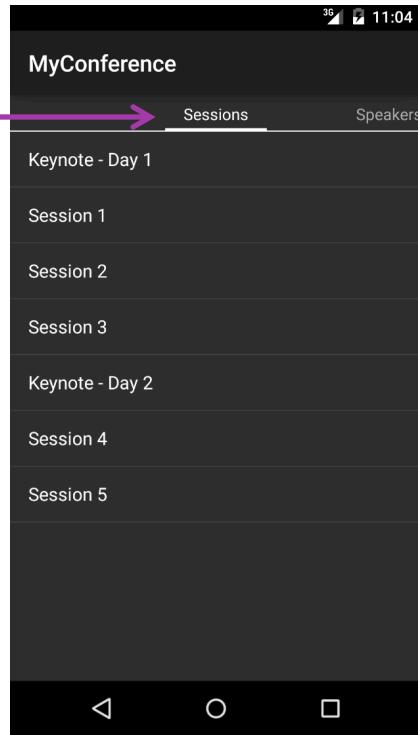
1. Display Fragments in a **ViewPager**
2. Show tabs using **PagerTabStrip**



Motivation

- ❖ Tabs let the user see their options and switch between functions quickly

Navigation tabs



Support Library

- ❖ The classes that implement Tab Navigation are in the v4 Support Library



Android Support Library v4 (21.0.3.0) [Details](#) [Remove](#)

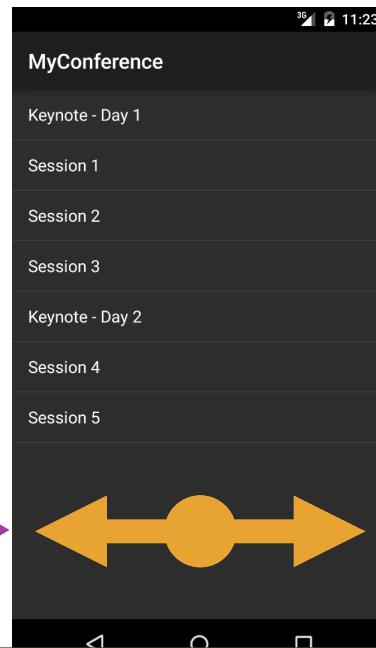
Provide backward-compatible versions of Android framework APIs.



What is **ViewPager**?

- ❖ **ViewPager** is a layout manager that lets the user step forward and back through a sequences of pages

Swipe gesture
used to change
pages →



← Does not have
built-in tabs

The Android docs say **ViewPager** is new so you should expect changes to the API.



How to use ViewPager

- ❖ Add a **ViewPager** to your layout file

ViewPager is a often the root node in your XML

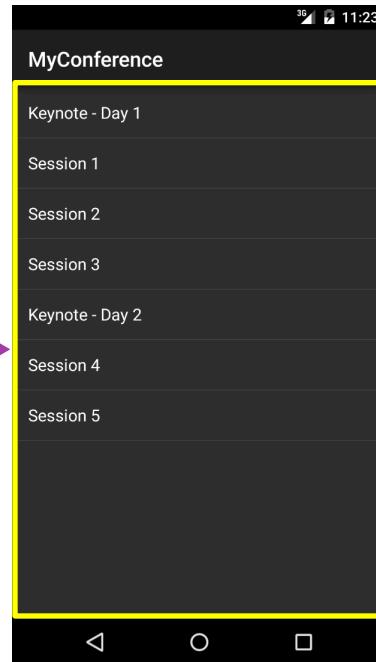
Pages are loaded via code-behind, not hardcoded in XML

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/viewPager"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</android.support.v4.view.ViewPager>
```

What are pages?

- ❖ The pages displayed by **ViewPager** are typically either Fragments or Views (we will use Fragments as they are more powerful and more common)

Typically a
Fragment



Fragment transactions

- ❖ **ViewPager** performs the fragment transactions for you, but you need to supply it with a **FragmentManager**

```
var fragment = new MyFragment();  
  
var transaction = base.FragmentManager.BeginTransaction();  
transaction.Replace(Resource.Id.myFrame, fragment);  
transaction.Commit();
```



ViewPager manages the Fragments for you,
you do not need to write this code

Fragment base type

- ❖ Fragments displayed by **ViewPager** must use the support-library **Fragment** class as their base

```
public class MyFragment : Android.Support.V4.App.Fragment  
{  
    ...  
}
```



Required because **ViewPager** uses the support version of **FragmentManager** for its fragment transactions

Activity base type

- ❖ Activities that host a **ViewPager** use the support-library **FragmentActivity** class as their base

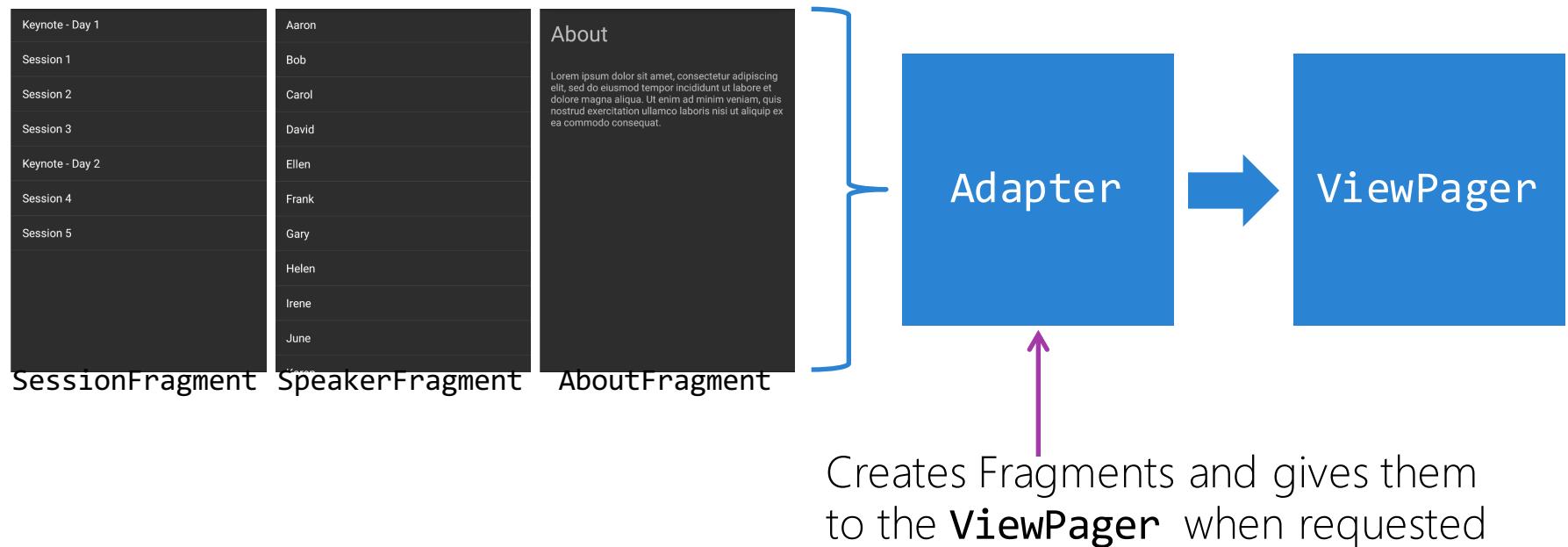
```
public class MainActivity : Android.Support.V4.App.FragmentActivity  
{  
    ...  
}
```



You inherit a **SupportFragmentManager** property that gives you the support version of the **FragmentManager** which **ViewPager** needs

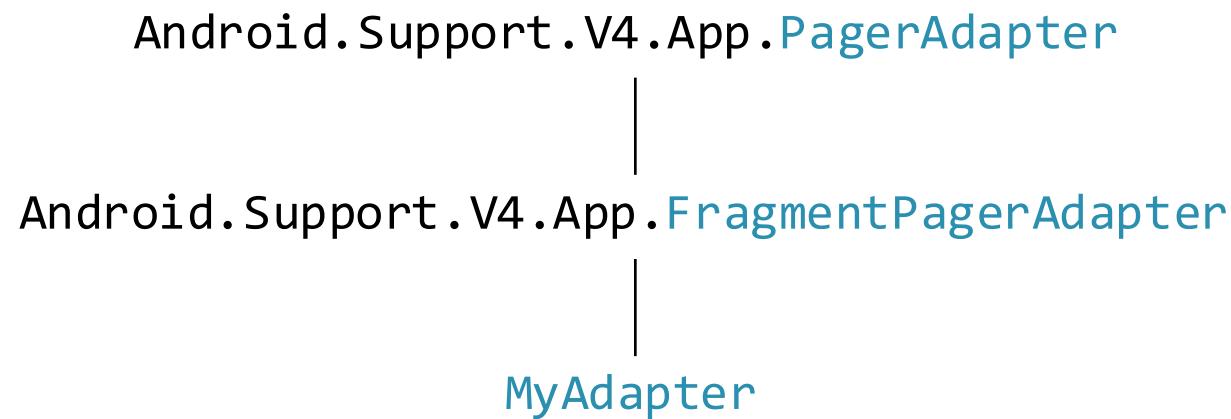
What is an adapter?

- ❖ Your adapter provides your pages to the **ViewPager**



Adapter base class

- ❖ You code an adapter that inherits from `FragmentPagerAdapter`



Derive from `FragmentStatePagerAdapter` when you have many pages.
It conserves memory by destroying Fragments that are not visible to the user.

Adapter FragmentManager

- ❖ You must pass a support **FragmentManager** to your adapter's base

```
public abstract class FragmentPagerAdapter : Android.Support.V4.View.PagerAdapter
{
    ...
    public FragmentPagerAdapter(Android.Support.V4.App.FragmentManager fm)
    {
        ...
    }
}
```



Your adapter's constructor needs to chain to its base constructor and pass the manager

Adapter implementation

- ❖ Your adapter provides the Fragments to the **ViewPager**

```
public class MyAdapter : Android.Support.V4.App.FragmentPagerAdapter
{
    Android.Support.V4.App.Fragment[] fragments;

    public MyAdapter(Android.Support.V4.App.FragmentManager fm, Android.Support.V4.App.Fragment[] fragments)
        : base(fm)
    {
        this.fragments = fragments;
    }

    public override int Count
    {
        get { return fragments.Length; }
    }

    public override Android.Support.V4.App.Fragment GetItem(int position)
    {
        return fragments[position];
    }
}
```

Number of Fragments →

Fragment at the given position →

Using an adapter

- ❖ You instantiate an adapter and load it into your **ViewPager**

```
protected override void OnCreate(Bundle bundle)
{
    var fragments = new Android.Support.V4.App.Fragment[]
    {
        new SessionFragment(),
        new SpeakerFragment(),
        new AboutFragment()
    };

    viewPager = FindViewById<Android.Support.V4.View.ViewPager>(Resource.Id.viewPager);

    viewPager.Adapter = new MyAdapter(base.SupportFragmentManager, fragments);
}
```

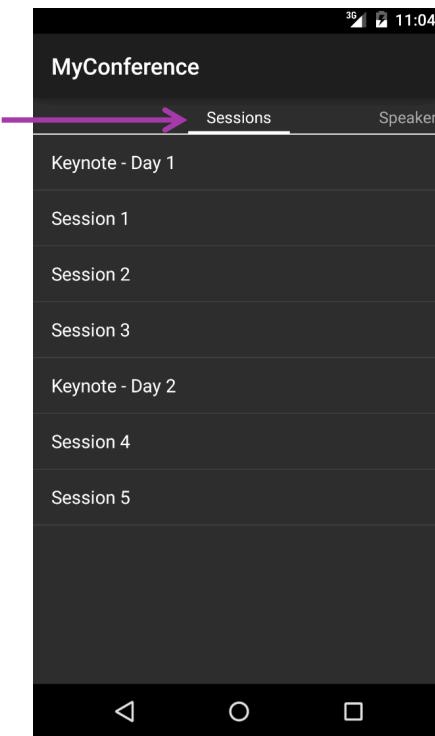
2. Assign

1. Create

What is PagerTabStrip?

- ❖ **PagerTabStrip** displays tabs inside a **ViewPager**

Current tab is highlighted and centered to give context



User can tap to navigate or swipe to scroll the tabs

How to use PagerTabStrip

- ❖ **PagerTabStrip** is intended for use with **ViewPager**, you just declare one as a child of your **ViewPager**

ViewPager will automatically find this since it is a child

```
<android.support.v4.view.ViewPager ... >  
  
    →<android.support.v4.view.PagerTabStrip  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:layout_gravity="top" /> ←  
  
</android.support.v4.view.ViewPager>
```

You choose tab position

Adapter tab titles

- ❖ You override a method in your adapter to provide the tab titles

```
public class MyAdapter : Android.Support.V4.App.FragmentPagerAdapter
{
    ...
    ICharSequence[] titles;

    public MyAdapter(... ICharSequence[] titles)
        : base(...)
    {
        this.titles = titles;
    }

    → public override ICharSequence GetPageTitleFormatted(int position)
    {
        return titles[position];
    }
}
```

Tab text at the
given position

Deprecated tab API

- ❖ Previous versions of Android provided a tabs API in the `ActionBar`

This method was deprecated in API level 21.



Using `ActionBar` for tabs is no longer the recommended practice



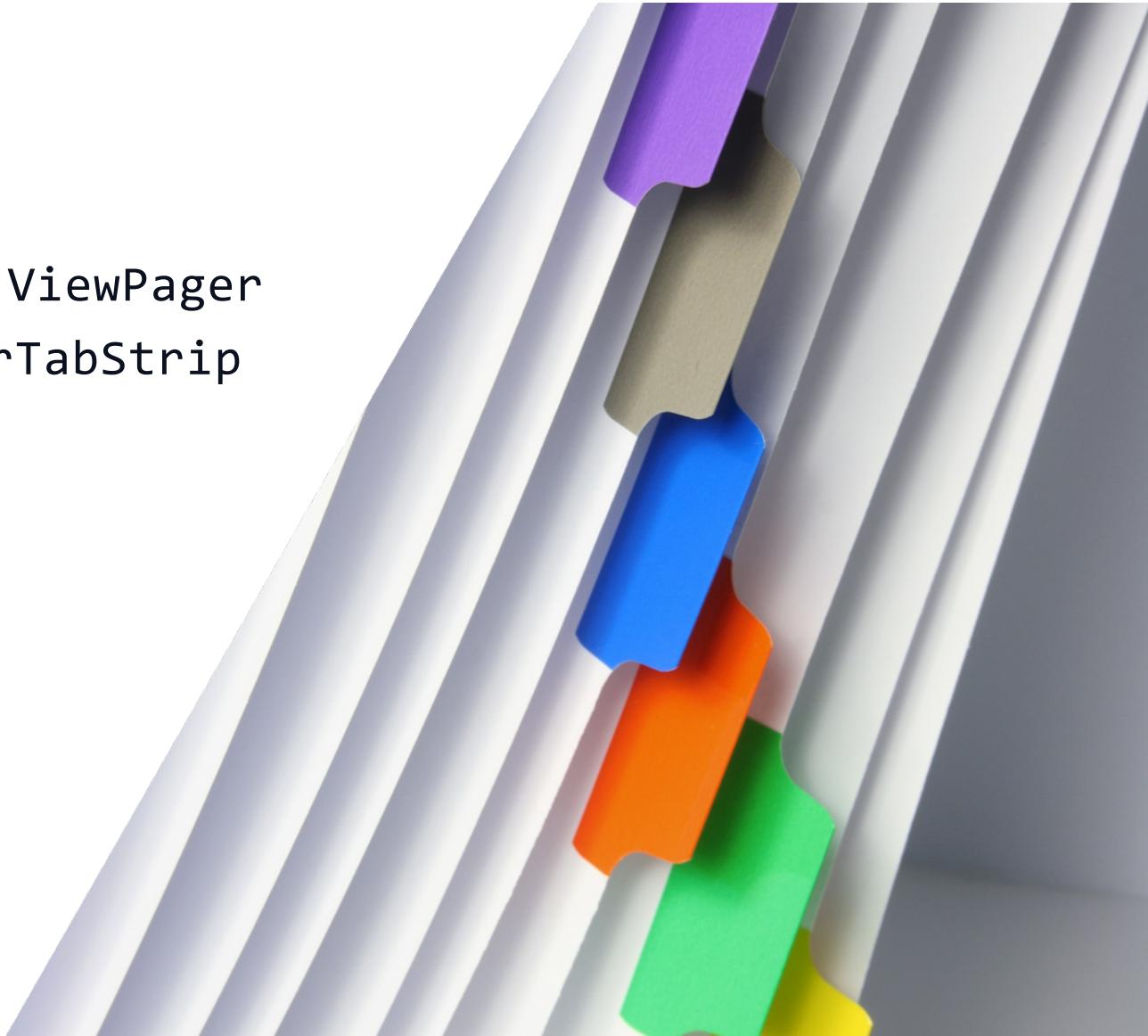
Individual Exercise

Tab Navigation



Summary

1. Display Fragments in a **ViewPager**
2. Show tabs using **PagerTabStrip**



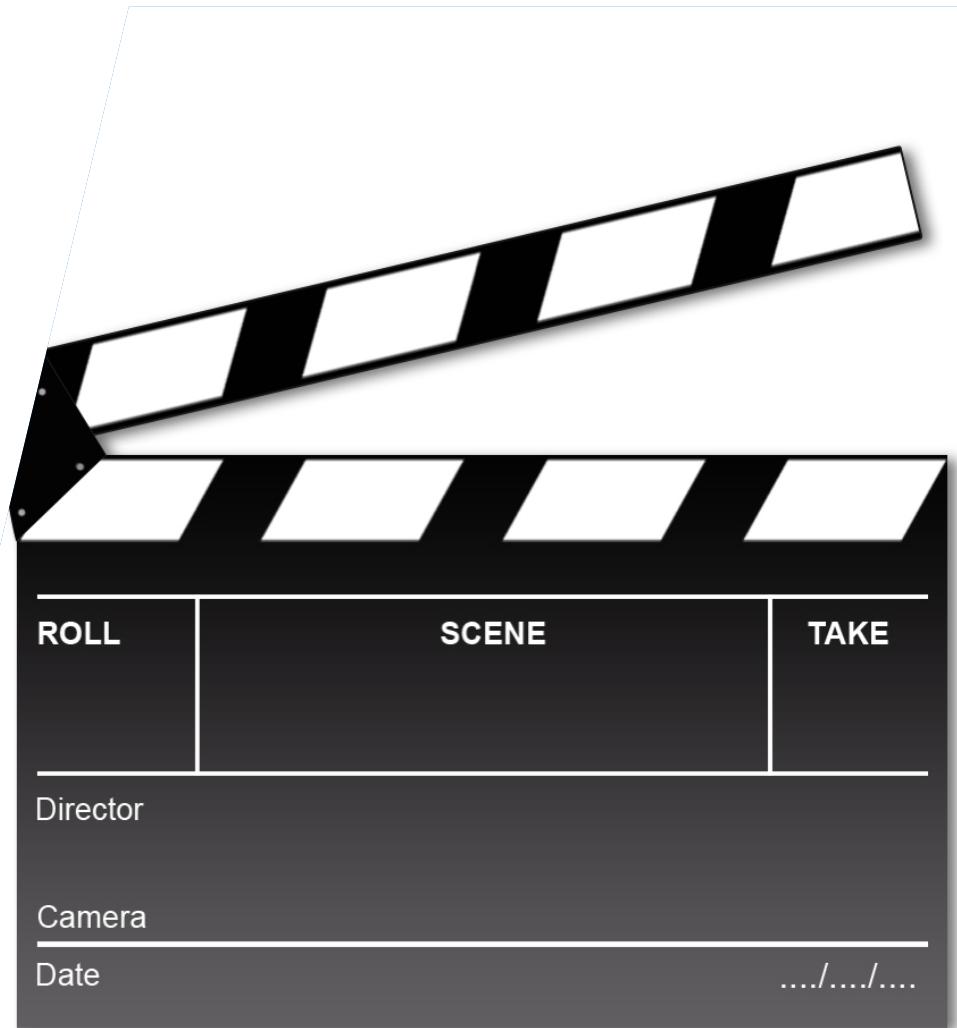


Introduce ActionBar



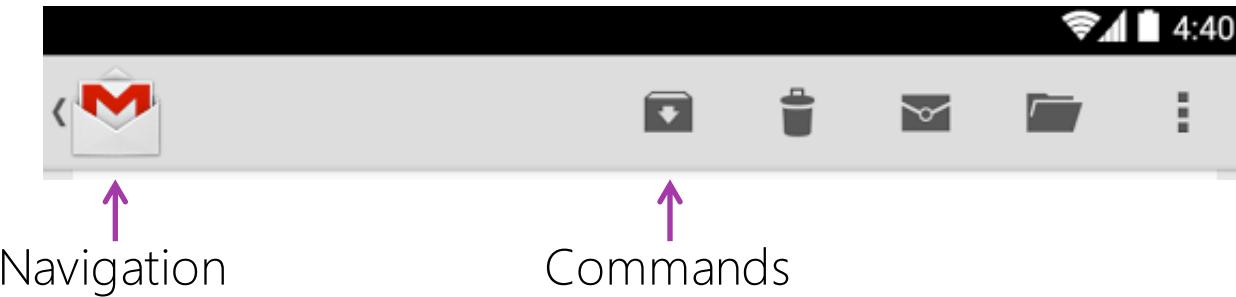
Tasks

1. Use an **ActionBar** in an Activity
2. Respond to item-click
3. Add a support library to your project



What is **ActionBar**?

- ❖ **ActionBar** is a control that hosts your app's navigation and command buttons



Navigation

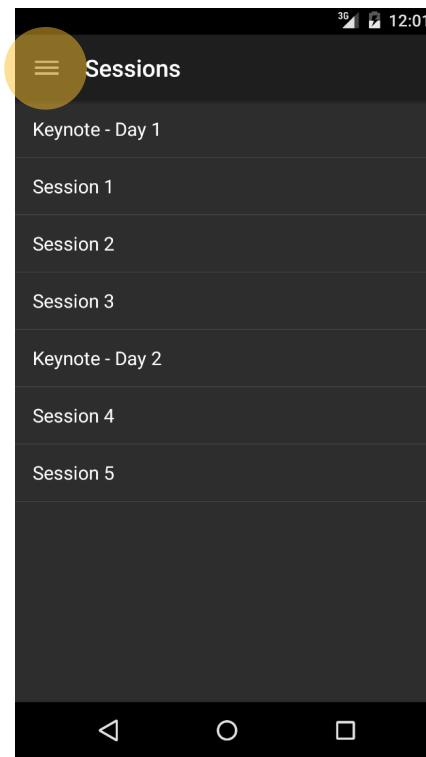
Commands



In this course, we will use **ActionBar** for navigation, not for commands.

Navigation controls in ActionBar

- ❖ Drawer navigation uses the Action Bar for the Drawer icon



ActionBar property

- ❖ The **Activity** class has an **ActionBar** property (added in API level 11)

```
public class Activity : ...
{
    ...
    public virtual ActionBar ActionBar { get; }
    ...
}
```



Every **Activity** has an **ActionBar**

ActionBar callback

- ❖ **ActionBar** item-click events are reported via an **Activity** method

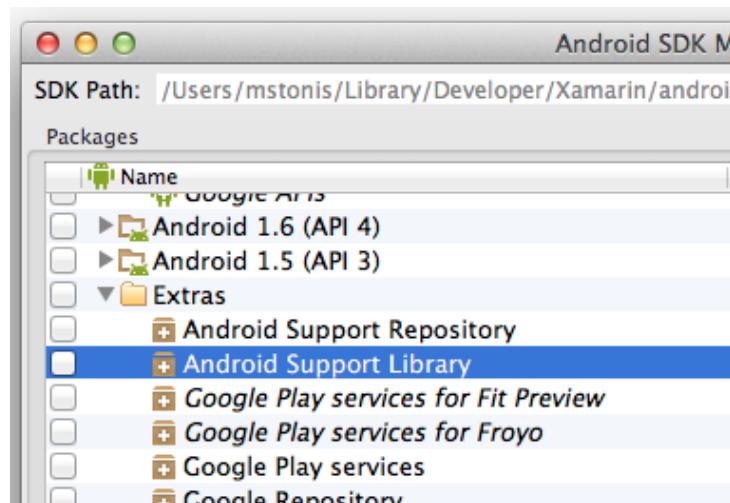
```
public class MainActivity : Activity
{
    ...
    public override bool OnOptionsItemSelected(IMenuItem item)
    {
        switch (item.ItemId)
        {
            ...
        }
    }
}
```



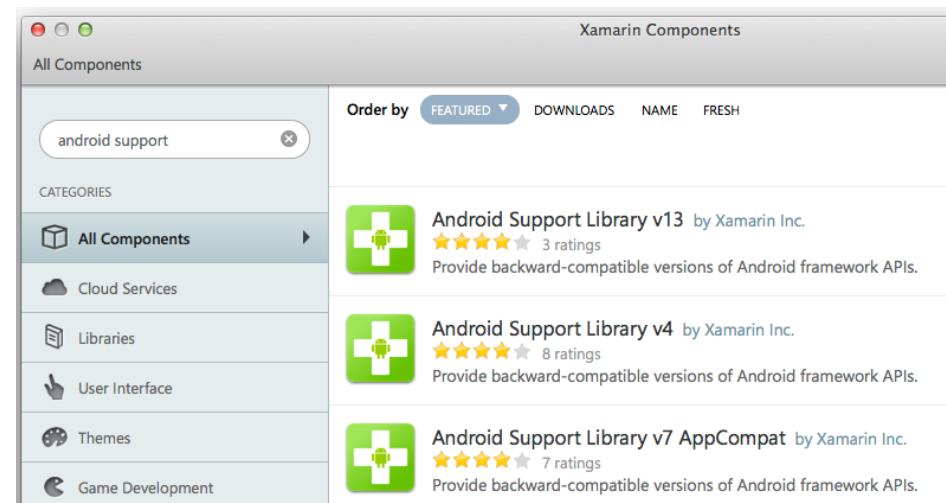
An identifier that indicates which element was clicked

Support Library

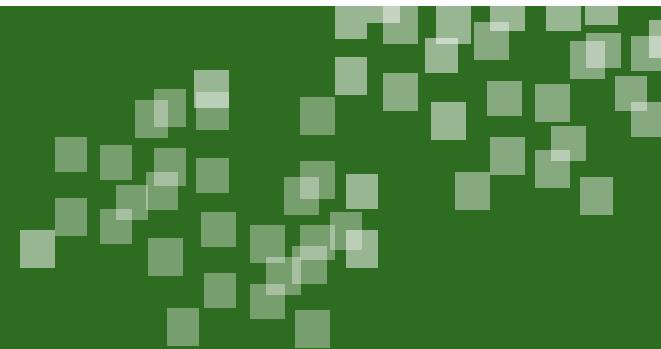
- ❖ The Action Bar is available in the Support Library and can be used back to version 2.1



1. Install on your dev machine



2. Add to your project



Flash Quiz



Flash Quiz

- ① **ActionBar** is only available in the latest version of Android.
- a) True
 - b) False
-

Flash Quiz

- ① **ActionBar** is only available in the latest version of Android.
- a) True
 - b) False
-

Flash Quiz

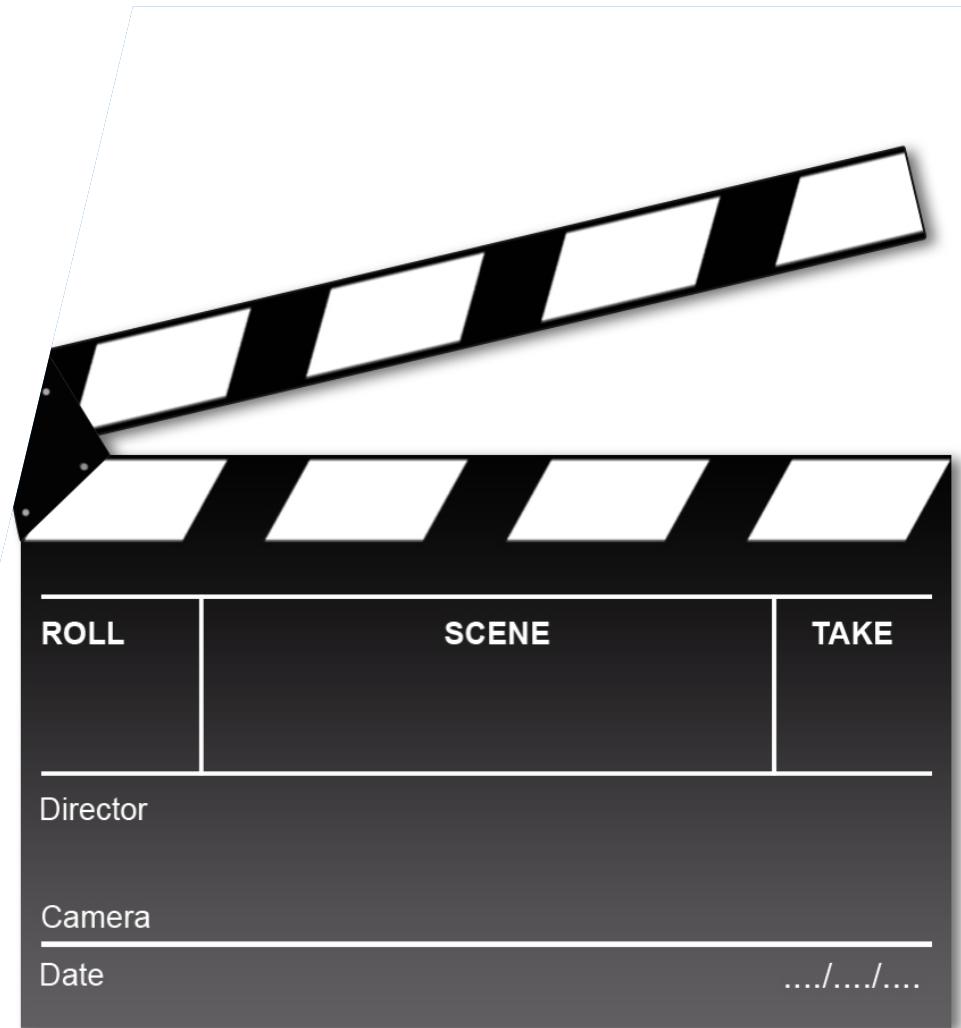
- ② What do you do to be notified of **ActionBar** item-clicks?
- a) Subscribe to the **ActionBar.ItemClick** event
 - b) Pass an **Action<int>** to the **ActionBar** constructor
 - c) Override an **Activity** method
-

Flash Quiz

- ② What do you do to be notified of **ActionBar** item-clicks?
- a) Subscribe to the `ActionBar.ItemClick` event
 - b) Pass an `Action<int>` to the `ActionBar` constructor
 - c) Override an **Activity** method
-

Summary

1. Use an **ActionBar** in an Activity
2. Respond to item-click
3. Add a support library to your project





Implement Drawer Navigation



Tasks

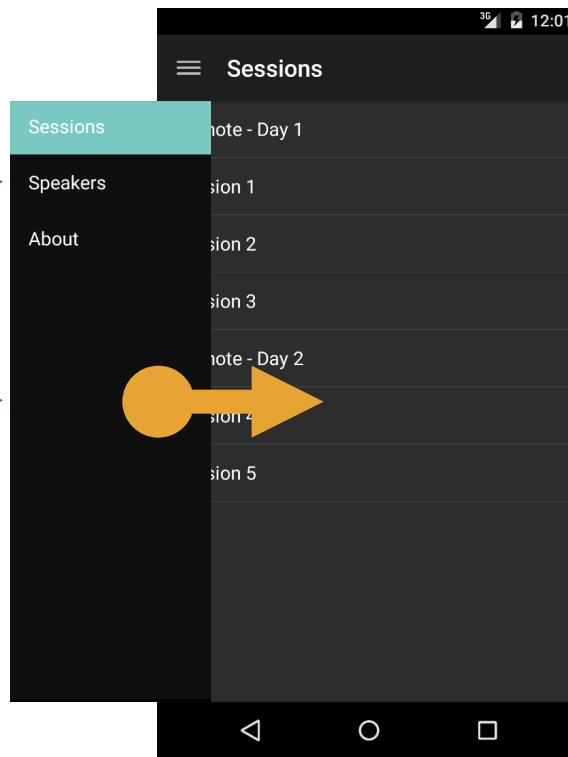
1. Code a **ListView** for your menu
2. Load a Fragment when your menu is clicked
3. Use a **DrawerLayout** to host your flyout menu
4. Use an **ActionBarDrawerToggle** to open/close the drawer



Motivation

- ❖ Drawer navigation works well when your app needs a large menu

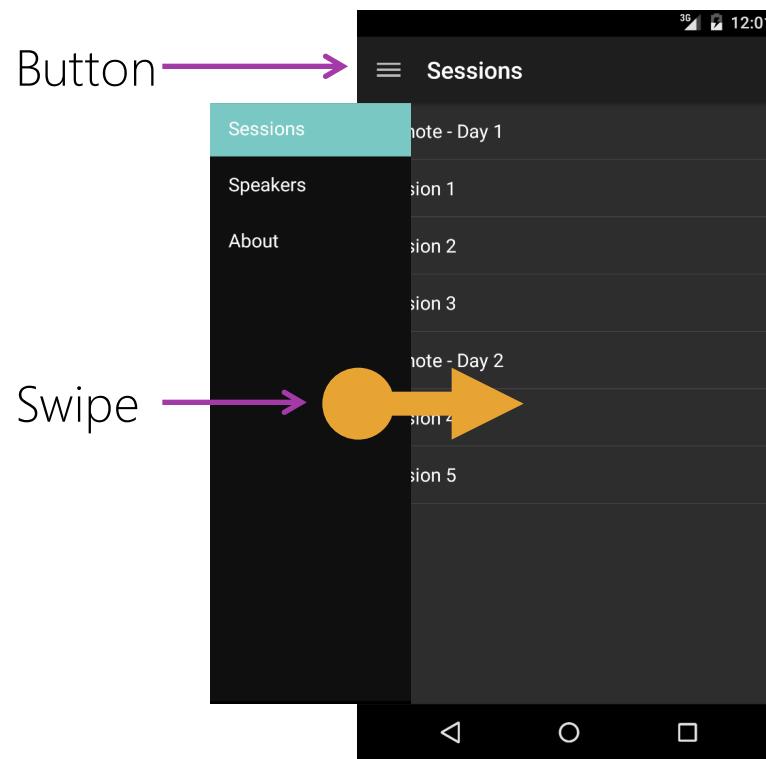
Menu has room
for many entries →



Menu overlays content
when open, but then
disappears when closed →

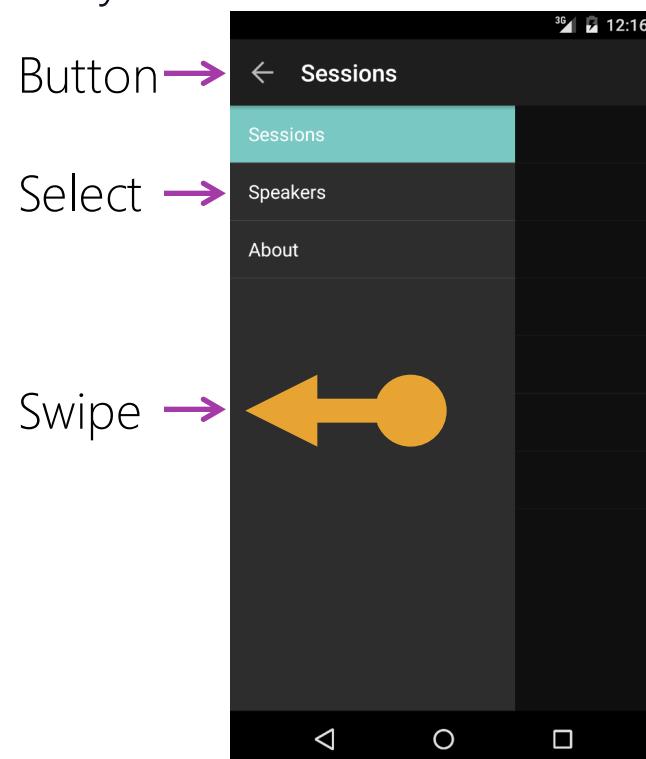
How to open the drawer?

- ❖ There are two ways for the user to open the drawer



How to close the drawer?

- ❖ There are several ways for the user to close the drawer

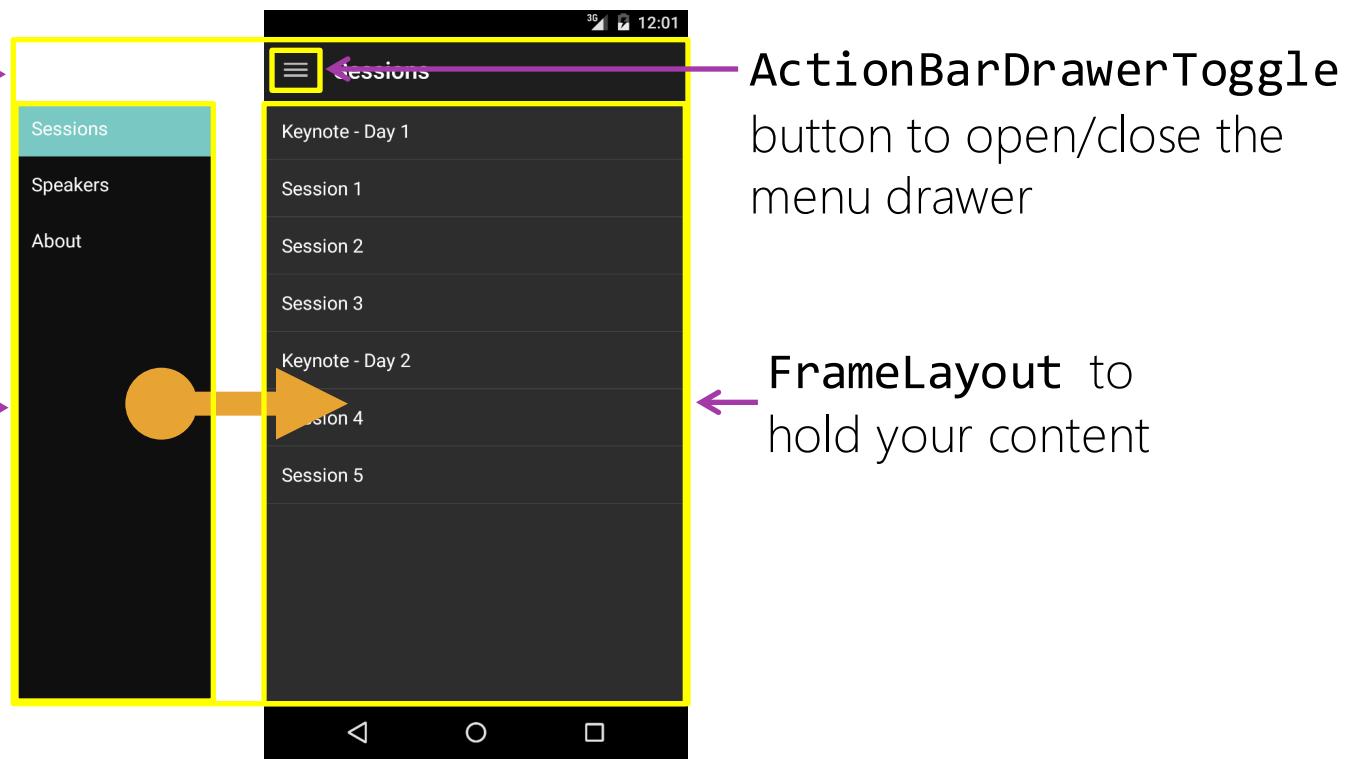


Architecture

- ❖ There are 4 pieces to the Drawer Navigation pattern

DrawerLayout
to hold all the parts

ListView
for the menu



Support Library

- ❖ The classes that implement Drawer Navigation are in the v4 and v7 Support Libraries (including v7 gets you v4 automatically)

DrawerLayout →



Android Support Library v4 (21.0.3.0) [Details](#) [Remove](#)

Provide backward-compatible versions of Android framework APIs.

ActionBarDrawerToggle →



Android Support Library v7 AppCompat (21.0.3.0) [Details](#) [Remove](#)

Provide backward-compatible versions of Android framework APIs.

Define your menu UI

- ❖ Typical to use a **ListView** for the drawer menu

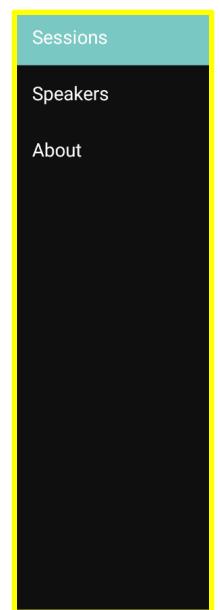
Navigation drawer goes on the left

You choose the width, should be 320dp or less

```
Main.axml
```

```
...
<ListView
    android:id="@+id/drawerListView"
    android:layout_gravity="start"
    android:choiceMode="singleChoice"
    android:layout_width="240dp"
    android:layout_height="match_parent"
    android:background="?android:attr/windowBackground" />
...

```



DrawerLayout uses the gravity setting to determine which child is the menu. You must set it to **start/left** for a left-side drawer or **DrawerLayout** will throw an exception.

Load your menu from code

- ❖ Typical to load your **ListView** menu from code-behind using an adapter

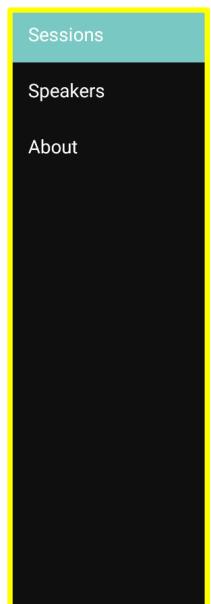
Menu
entries

```
MainActivity.cs
```

```
string[] titles = new string[] { "Sessions", "Speakers", "About" };

drawerListView = FindViewById<ListView>(Resource.Id.drawerListView);

drawerListView.Adapter = new ArrayAdapter<string>
(
    this,
    Resource.Layout.ListViewMenuRow, // layout file for row
    Resource.Id.menuRowTextView,    // id of TextView in layout file
    titles
);
```



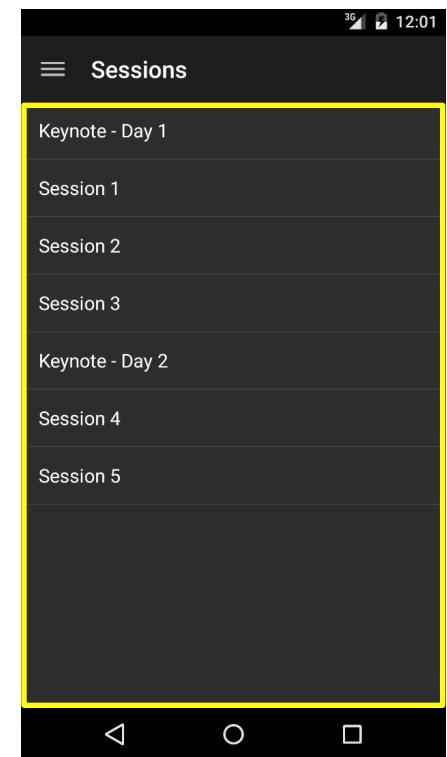
Define your content UI

- ❖ Use a **FrameLayout** to host your content

Will hold
Fragments
loaded
from code

...
**<FrameLayout**
 android:id = "@+id/frameLayout"
 android:layout_width = "match_parent"
 android:layout_height= "match_parent" />
...

Occupies the entire UI area



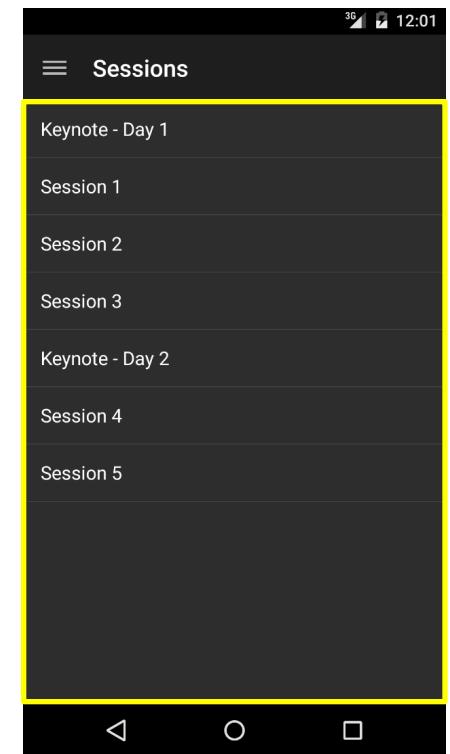
Load your content from code

- ❖ Update the **FrameLayout** when the user selects from the menu

```
drawerListView.ItemClick += OnMenuItemClick;           MainActivity.cs  
  
void OnMenuItemClick(object sender, AdapterView.ItemClickEventArgs e)  
{  
    base.FragmentManager  
    .BeginTransaction()  
    .Replace(Resource.Id.frameLayout, fragments[e.Position])  
    .Commit();  
  
    this.Title = titles[e.Position];  
}
```

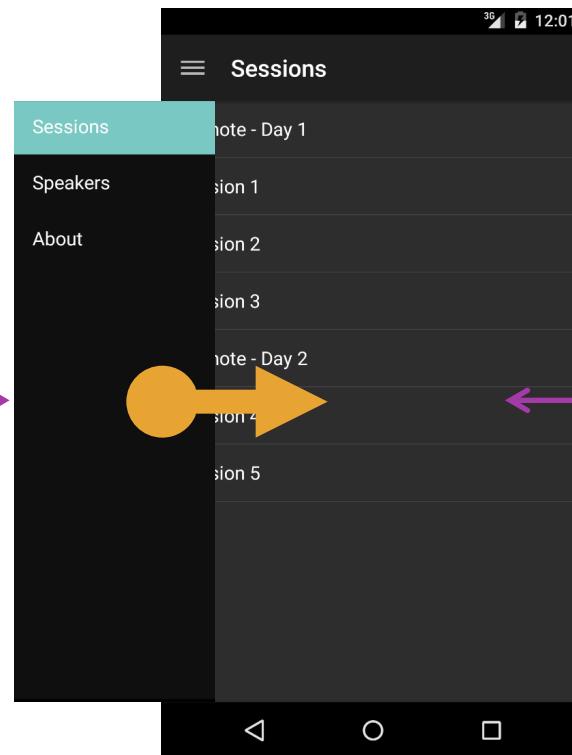


Load a Fragment based on the
ListView item the user clicked



What is DrawerLayout?

- ❖ **DrawerLayout** is a layout manager that provides a flyout menu



It implements the
"sliding drawer"
containing your list

It is a layout manager
so it has children that
it displays for you

Define your root layout UI

- ❖ Use a **DrawerLayout** as your root layout manager

Main.axml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout ...>

    Content → <FrameLayout ... />

    Menu → <ListView ... />

</android.support.v4.widget.DrawerLayout>
```

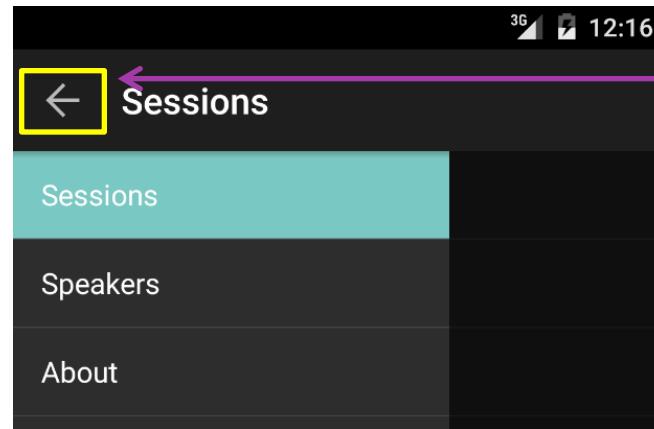
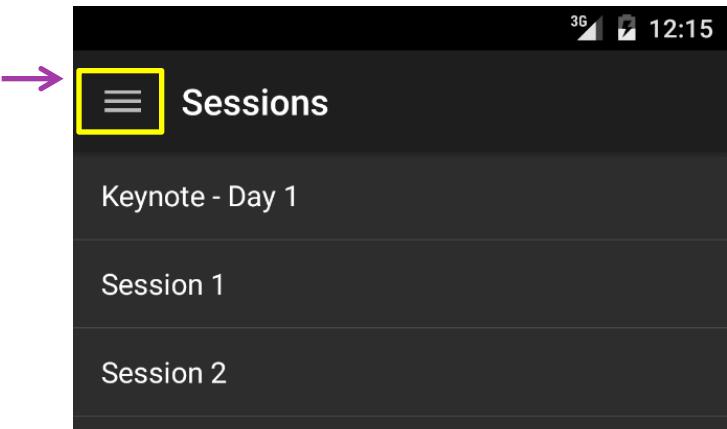


Android says to define the **FrameLayout** first and the **ListView** second so the menu is rendered above the content in Z-order.

What is `ActionBarDrawerToggle`?

- ❖ `ActionBarDrawerToggle` is a toggle button that is hosted inside your `ActionBar` and used to open/close a menu drawer

Looks like
this when
the menu
is closed



Animates
to this when
the menu
is open

ActionBarDrawerToggle creation

- ❖ You instantiate an `ActionBarDrawerToggle` in code, not XML

```
public class MainActivity : Activity
{
    ...
    ActionBarDrawerToggle drawerToggle;

    protected override void OnCreate(Bundle bundle)
    {
        ...
        Allocate → drawerToggle = new ActionBarDrawerToggle(...);
        ...
    }
}
```

ActionBarDrawerToggle accessibility

- ❖ `ActionBarDrawerToggle` requires two string resources for use by accessibility services

Resources/values/Strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="DrawerOpenDescription">Open navigation drawer</string>
    <string name="DrawerCloseDescription">Close navigation drawer</string>
</resources>
```

Pass your accessibility
strings to the constructor

```
drawerToggle = new ActionBarDrawerToggle
(
    ...,
    Resource.String.DrawerOpenDescription,
    Resource.String.DrawerCloseDescription
);
```

ActionBar integration [UI]

- ❖ `ActionBarDrawerToggle` adds itself to your `ActionBar` automatically, you just need to enable your `ActionBar`'s Home button

`MainActivity.cs`

```
protected override void OnCreate(Bundle bundle)
{
    ...
    drawerToggle = new ActionBarDrawerToggle(this, ...);

    ActionBar.SetDisplayHomeAsUpEnabled(true);
    ...
}
```

It will replace the
normal Home
icon with its own

You must pass in your `Activity`, which
allows it to access your `ActionBar`

ActionBar integration [clicks]

- ❖ You send ActionBar-click events to the `ActionBarDrawerToggle` or it won't know when the user clicks it

MainActivity.cs

It verifies the Home button was clicked and then toggles the drawer

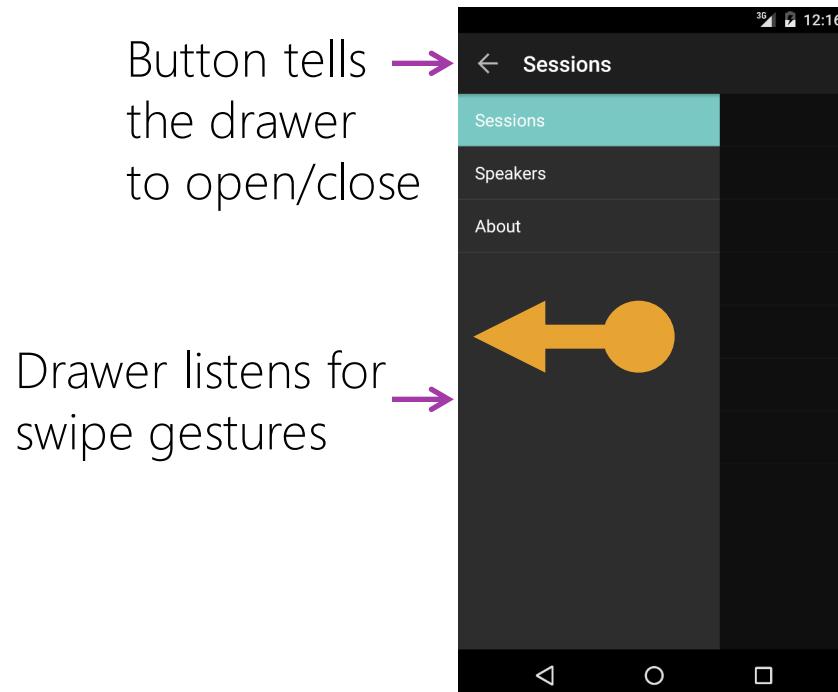
Handle clicks on other `ActionBar` buttons

```
public override bool OnOptionsItemSelected(IMenuItem item)
{
    if (drawerToggle.OnOptionsItemSelected(item))
        return true;

    switch (item.ItemId)
    {
        ...
    }
}
```

Drawer management [overview]

- ❖ **ActionBarDrawerToggle** and **DrawerLayout** collaborate to manage the drawer



Drawer management [automatic]

- ❖ **ActionBarDrawerToggle** and **DrawerLayout** have references to each other so they can work together

MainActivity.cs

```
protected override void OnCreate(Bundle bundle)
{
    ...
    drawerLayout = FindViewById<DrawerLayout>(Resource.Id.drawerLayout);

    drawerToggle = new ActionBarDrawerToggle(this, drawerLayout, ...);

    drawerLayout.SetDrawerListener(drawerToggle);
    ...
}
```

Drawer reports changes to the button so the button can toggle its icon

Button can call open/close methods on the drawer when the user clicks on it

Drawer management [manual]

- ❖ You have to close the drawer when the user selects your menu item

MainActivity.cs

```
void OnMenuItemClick(object sender, AdapterView.ItemClickEventArgs e)
{
    base.FragmentManager
        .BeginTransaction()
        .Replace(Resource.Id.frameLayout, fragments[e.Position])
        .Commit();

    this.Title = titles[e.Position];
}

drawerLayout.CloseDrawer(drawerListView);
```

Your code
closes the
drawer

Identifies which drawer to close (needed because
DrawerLayout can have left and right drawers)

Drawer↔button sync at startup

- ❖ After your Activity's initialization is finished, you need to tell the toggle button to synchronize its icon with the open/closed state of the drawer

Android recommends calling it here, after all Activity setup/restore is complete

MainActivity.cs

```
public class MainActivity : Activity
{
    ...
    protected override void OnPostCreate(Bundle savedInstanceState)
    {
        drawerToggle.SyncState();

        base.OnPostCreate(savedInstanceState);
    }
}
```



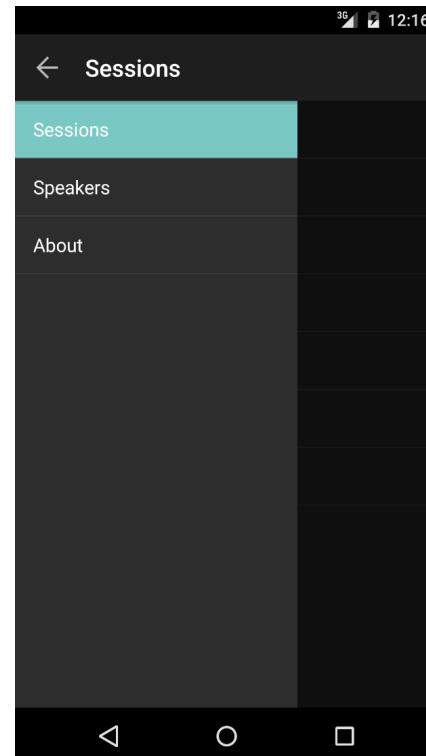
Group Exercise

Drawer Navigation



Drawer guidelines

- ❖ Android has design guidelines to help you code Navigation Drawer
 - Start with the drawer open on first run
 - Make your drawer listings descriptive with icons, images, and counters



 Our examples could not show all the best practices, please read the guidelines:
<https://developer.android.com/design/patterns/navigation-drawer.html>

Summary

1. Code a **ListView** for your menu
2. Load a Fragment when your menu is clicked
3. Use a **DrawerLayout** to host your flyout menu
4. Use an **ActionBarDrawerToggle** to open/close the drawer



Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

