

AND180



## Toolbar and App Bar

Download class materials from  
[university.xamarin.com](http://university.xamarin.com)



Xamarin University

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.

Microsoft or 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 Microsoft or Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.

© 2014-2017 Xamarin Inc., Microsoft. All rights reserved.

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

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

# Objectives

- ❖ Add a **Toolbar** to an Activity
- ❖ Populate **Toolbar** Actions from an XML menu file
- ❖ Use a **Toolbar** as an Activity's app bar
- ❖ Set a navigation icon on an app bar





# Add a Toolbar to an Activity

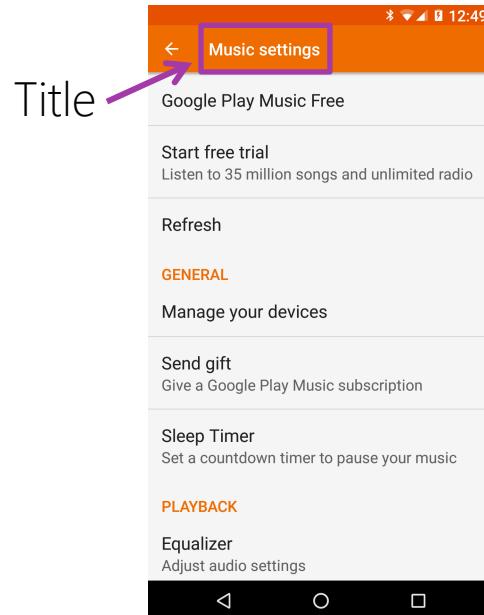
# Tasks

1. Add a **Toolbar** to your UI
2. Set text and logo



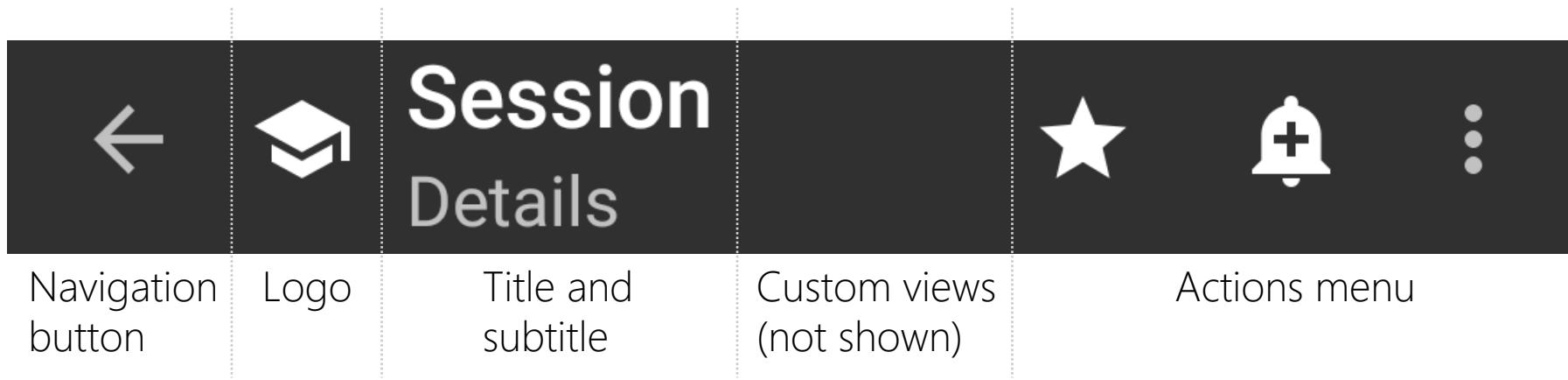
# Motivation

- ❖ Activities often contain a title and/or logo for branding and to indicate the user's current location within the app



# What is Toolbar?

- ❖ **Toolbar** is a horizontal menu that you position anywhere in your UI – it has several standard elements that you can use or ignore as needed



# Packaging

- ❖ **Toolbar** is available in two places

## Toolbar

```
public class Toolbar  
extends ViewGroup  
  
java.lang.Object  
↳ android.view.View  
    ↳ android.view.ViewGroup  
        ↳ android.widget.Toolbar
```

Standard in Android  
API level 21 and higher

## Toolbar

```
public class Toolbar  
extends ViewGroup  
  
java.lang.Object  
↳ android.view.View  
    ↳ android.view.ViewGroup  
        ↳ android.support.v7.widget.Toolbar
```

v7 Support Library for use  
on older Android versions



We will use the Support Library version for maximum compatibility

# Theme dependency

- ❖ Apps using the Support **Toolbar** must select one of the AppCompat themes

AndroidManifest.xml

```
<application
    ...
    android:theme="@style/Theme.AppCompat">
</application>
```



Needed when using  
the Support **Toolbar**

# How to create a Toolbar

- ❖ It is typical to use XML to add a **Toolbar** to your layout

```
<LinearLayout ...>  
  
    <android.support.v7.widget.Toolbar ... />  
  
</LinearLayout>
```



Include the package name since we are using the **Toolbar** from the Support library

# No Android namespace

- ❖ The Support **Toolbar** defines several XML attributes – since they come from a library rather than standard Android, they are **not** in the Android namespace

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" ...>  
  
    <android.support.v7.widget.Toolbar android:title="Session" ... />  
    ...
```



Toolbar's **title** attribute  
will not be found, this setting  
will be ignored at runtime

# App namespace

- ❖ Support Toolbar attributes are in your app's namespace – use the special symbol **res-auto** to define an XML prefix to reach them

The prefix **app** is used by convention

Replaced with your app's package name

```
<LinearLayout xmlns:app="http://schemas.android.com/apk/res-auto" ...>
```

```
    <android.support.v7.widget.Toolbar app:title="Session" ... />
```

```
    ...
```

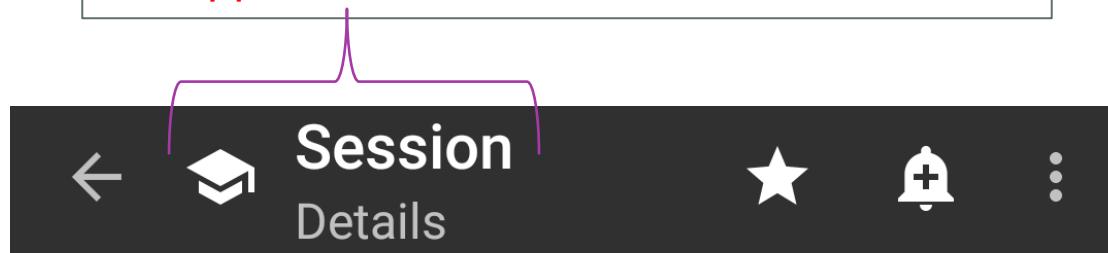


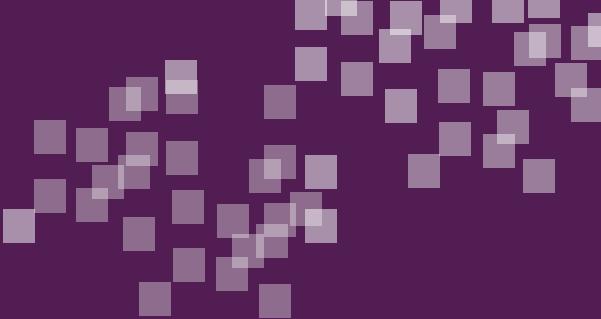
This will correctly locate Toolbar's **title** attribute

# How to set logo and text

- ❖ It is typical to use XML attributes to set the **Toolbar**'s logo and text

```
<android.support.v7.widget.Toolbar  
    app:logo="@drawable/hat"  
    app:title="Session"  
    app:subtitle="Details" ... />
```





# Individual Exercise

Add a Toolbar to an Activity

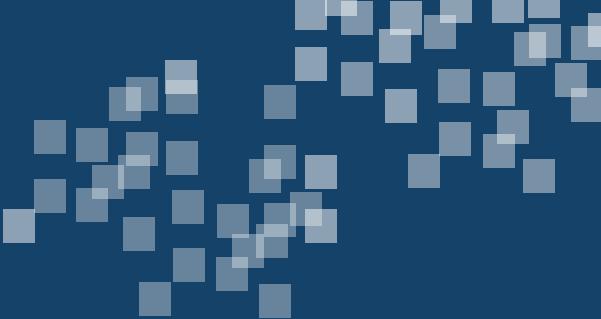


Xamarin  
University

# Summary

1. Add a **Toolbar** to your UI
2. Set text and logo





# Populate Toolbar Actions from an XML menu file

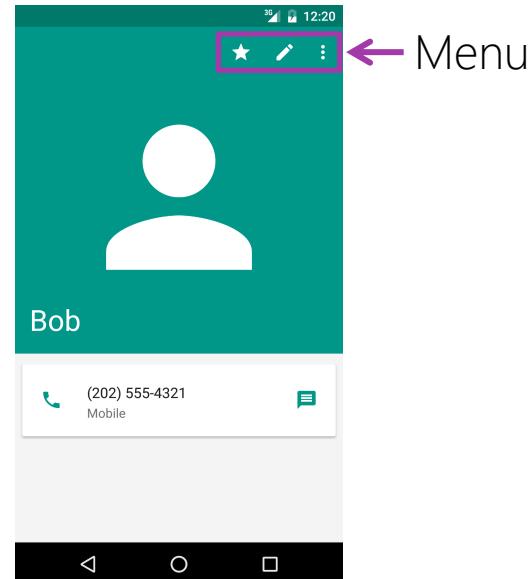
# Tasks

1. Define an XML menu file
2. Inflate the XML menu file into a **Toolbar**'s actions menu
3. Create hierarchical menus
4. Create checkable menu items



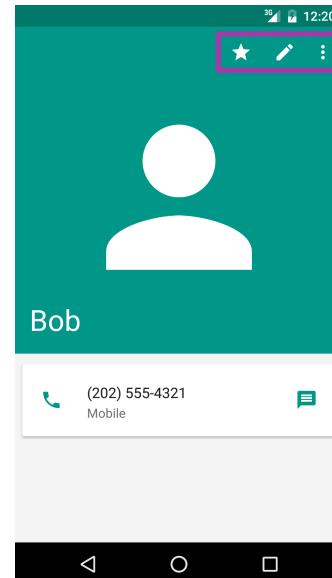
# Motivation

- ❖ Many Activities give the user a menu for easy access to common commands



# What are Actions?

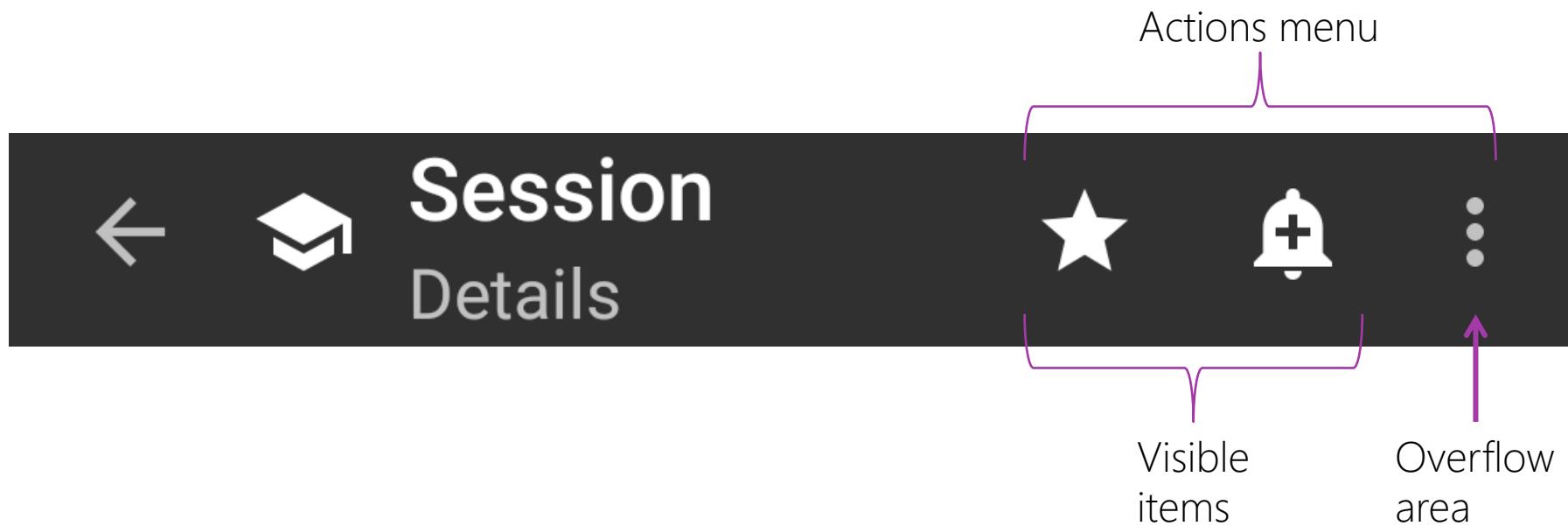
- ❖ *Actions* are the most-useful commands a user will need on a page – Android uses the acronym FIT (Frequent, Important, or Typical) to describe what qualifies as an action



← Actions for a contact include  
*Add to Favorites* and *Edit*

# Toolbar's actions menu

- ❖ **Toolbar** has an actions menu built in



# What is a Menu Item?

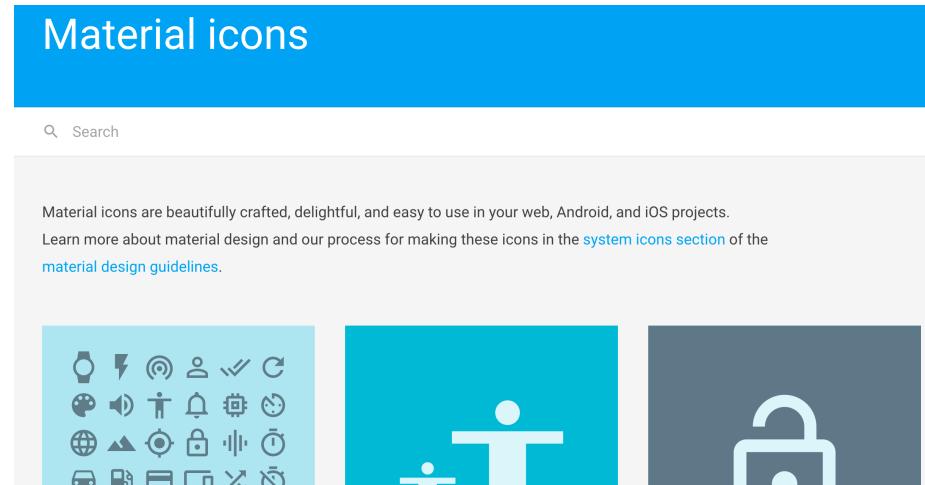
- ❖ `IMenuItem` represents an entry on an Android menu

```
public interface IMenuItem : IJavaObject, IDisposable
{
    ...
    Text → IMenuItem SetTitle (int title);
    Image → IMenuItem SetIcon (Drawable icon);
    Positioning → void SetShowAsAction (ShowAsAction actionEnum);
    Hierarchy → ISubMenu SubMenu { get; }
    Checked state → IMenuItem SetChecked(bool value);
}
```



# Icons

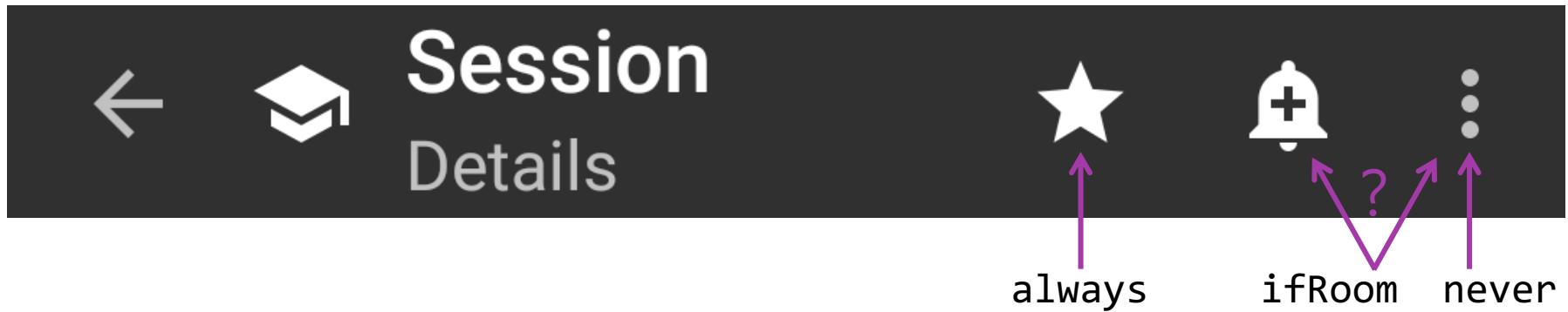
- ❖ Google provides many graphics appropriate for use as action icons at <https://design.google.com/icons/>



If you create your own icons, you should provide multiple sizes so they look good on all devices, see: [https://developer.android.com/guide/practices/ui\\_guidelines/icon\\_design\\_action\\_bar.html](https://developer.android.com/guide/practices/ui_guidelines/icon_design_action_bar.html)

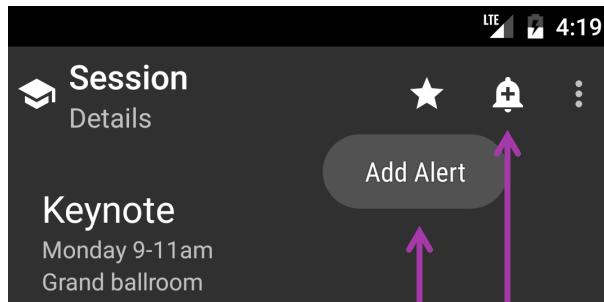
# Menu item location

- ❖ The **showAsAction** property determines whether menu items are placed directly on the toolbar or in the overflow area

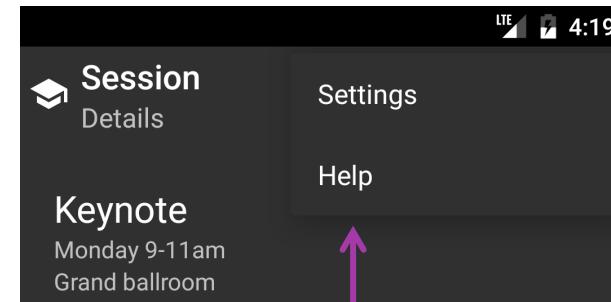


# Menu item display

- ❖ Menu items are displayed differently depending on location



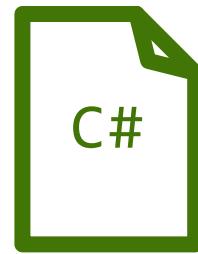
Title used  
as item's  
tooltip      Icon used  
when item  
is visible



Only the title  
is used in the  
overflow area

# Ways to create menus

- ❖ You can create menu items either in code or XML



Rare, but useful if  
menu content needs  
to be dynamic



Concise and follows the  
standard Android paradigm  
to define UI in markup

# Create menu items in code

- ❖ Use **Toolbar**'s **Menu** property and its **Add** method to create and add new menu items

```
var toolbar = FindViewById<Toolbar>(Resource.Id.toolbar);  
  
Text → IMenuItem item = toolbar.Menu.Add(Resource.String.add_alert);  
  
Image → item.SetIcon(Resource.Drawable.ic_add_alert_white_24dp);  
  
Positioning → item.SetShowAsAction(ShowAsAction.IfRoom);
```

# Create menu items in XML

- ❖ To create a menu in XML, first define a menu file and then use code to inflate it

Define the  
menu items →

Resources/menu/actions.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu ...>
  ...
</menu>
```

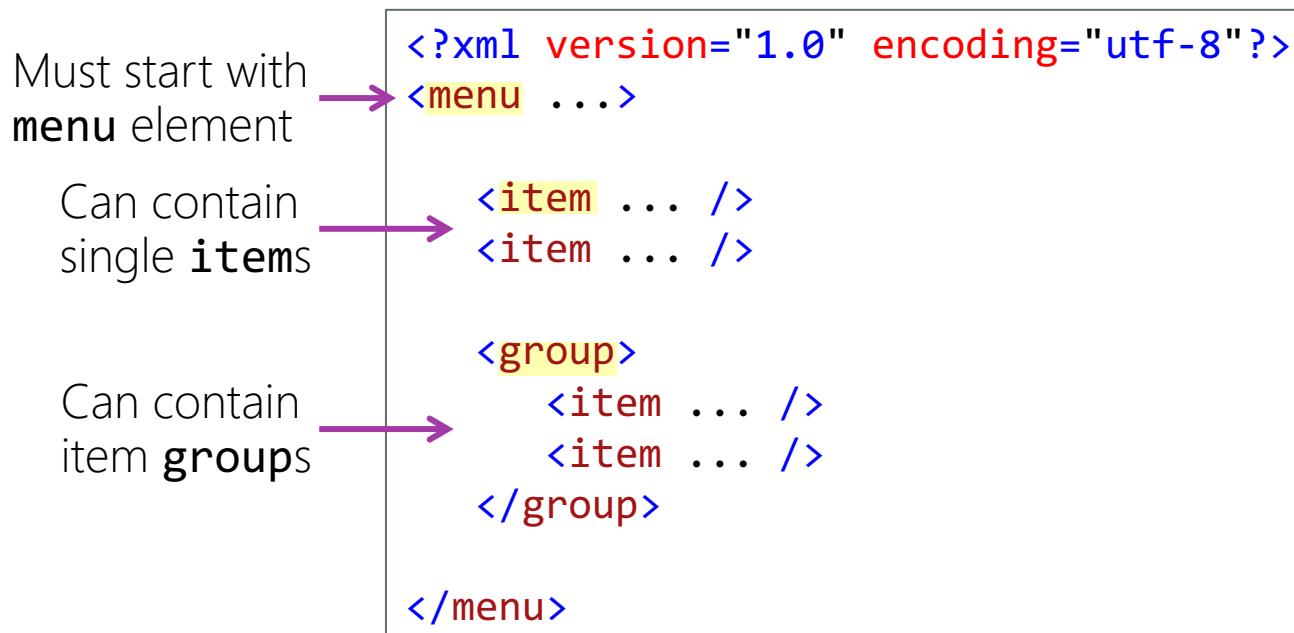
File placed  
in the menu  
folder

Inflate →

```
var toolbar = FindViewById<Toolbar>(Resource.Id.toolbar);
toolbar.InflateMenu(Resource.Menu.actions);
```

# Menu file structure

- ❖ A menu resource file defines a menu using XML  
(See <https://developer.android.com/guide/topics/resources/menu-resource.html>)



```
<?xml version="1.0" encoding="utf-8"?>
<menu ...>
    <item ... />
    <item ... />

    <group>
        <item ... />
        <item ... />
    </group>

</menu>
```

Must start with **menu** element

Can contain single **items**

Can contain item **groups**

# What is the <menu> element?

- ❖ The **menu** element is the root of a menu file – it defines any needed namespace prefixes but has no other attributes

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    ...
</menu>
```

The res-auto prefix is  
needed when using  
the support Toolbar

The standard  
Android prefix  
is always needed

# What is the <item> element?

- ❖ The **item** element defines a menu entry in XML

Add an **id** to any item you will need to identify from code

```
<item  
    android:id      = "@+id/addToFavorites"  
    android:title   = "Add To Favorites"  
    android:icon    = "@drawable/ic_grade_white_24dp"  
    app:showAsAction= "always"  
    ...  
/>
```

Text → android:title  
Image → android:icon  
Positioning → app:showAsAction="always"

Use the **res-auto** prefix for **showAsAction**  
when this item is inside a Support Toolbar

# Menu item click event

- ❖ Toolbar has a **MenuItemClick** event – use the item id to determine which menu entry was clicked

```
var toolbar = FindViewById<Toolbar>(Resource.Id.toolbar);
toolbar.MenuItemClick += OnClick;
```

Detect  
which  
item was  
clicked

```
void OnClick(object sender, Toolbar.MenuItemEventArgs e)
{
    → switch (e.Item.ItemId)
    {
        case Resource.Id.addAlert: ... break;
        case Resource.Id.addToFavorites: ... break;
    }
}
```

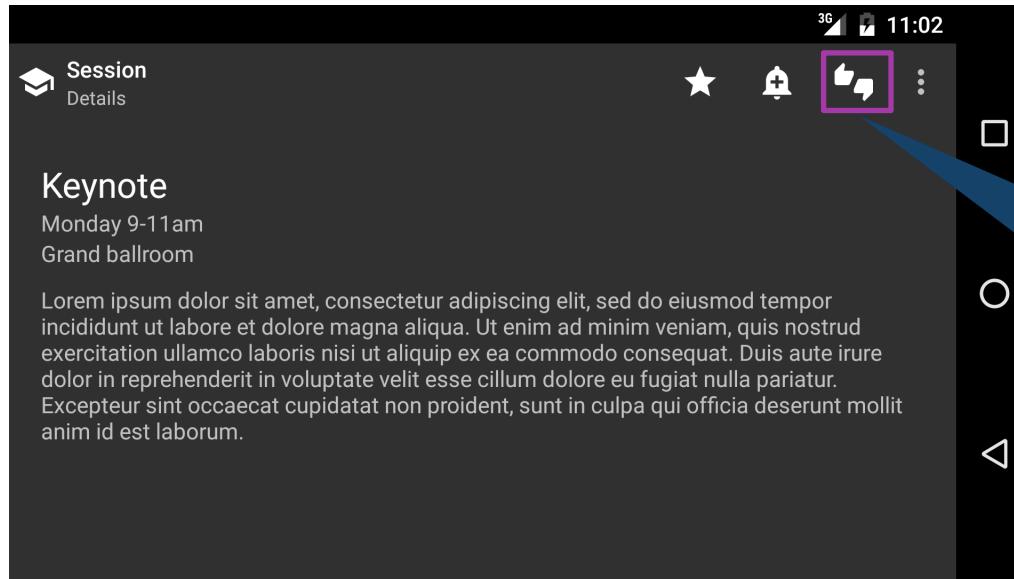


# Individual Exercise

Populate Toolbar Actions from an XML menu file

# Motivation [hierarchy]

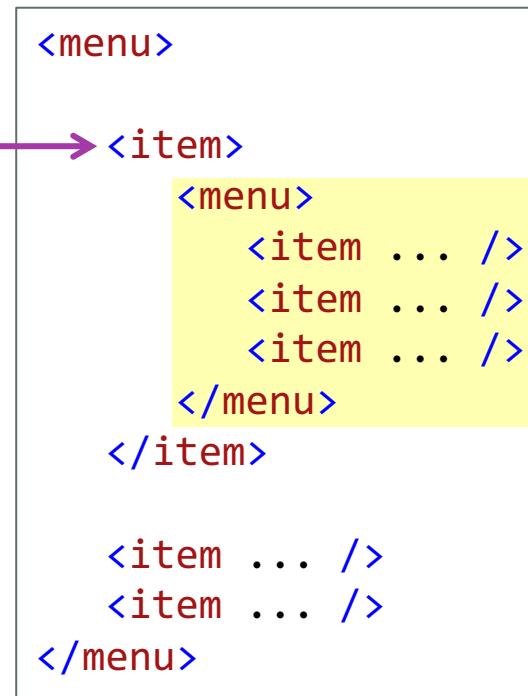
- ❖ Menus can be hierarchical – a top-level entry may lead to a sub-menu



# What is a sub-menu?

- ❖ A sub-menu is a `menu` contained within a `item`

Each `item` can  
optionally contain  
one `menu`



```
<menu>
    <item>
        <menu>
            <item ... />
            <item ... />
            <item ... />
        </menu>
    </item>

    <item ... />
    <item ... />
</menu>
```

# Submenu behavior

- ❖ An **item**'s sub-menu is hidden until the user selects the **item**

When the

user selects this item...

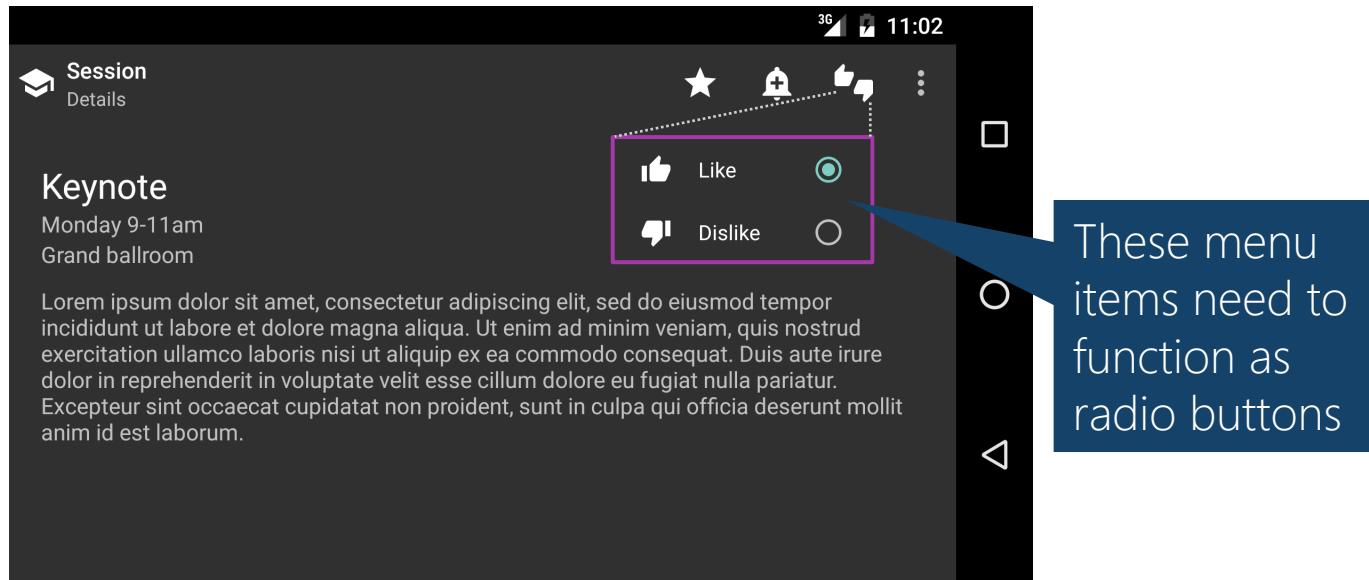
...this menu

appears

```
→<item android:id="@+id/rating" ...>
    <menu>
        →<item android:id="@+id/like" ... />
        <item android:id="@+id/dislike" ... />
    </menu>
</item>
```

# Motivation [checkable]

- ❖ Menu items that let you select one option from several choices need to support "checkable" behavior



# What is the <group> element?

- ❖ The **group** element is a container for a collection of **items**

A group containing two items

```
<group>
    <item
        android:id      ="@+id/like"
        android:icon   ="@drawable/ic_thumb_up_white_24dp"
        android:title="Like" />

    <item
        android:id      ="@+id/dislike"
        android:icon   ="@drawable/ic_thumb_down_white_24dp"
        android:title="Dislike" />

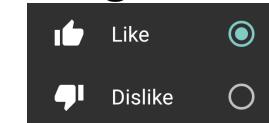
</group>
```

# Checkable <group>

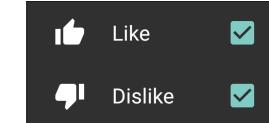
- ❖ The **group** element provides checkable behavior for the items it holds

```
<group android:checkableBehavior="single">  
  
    <item  
        android:id      ="@+id/like"  
        android:icon   ="@drawable/ic_thumb_up_white_24dp"  
        android:title="Like" />  
  
    <item  
        android:id      ="@+id/dislike"  
        android:icon   ="@drawable/ic_thumb_down_white_24dp"  
        android:title="Dislike" />  
  
</group>
```

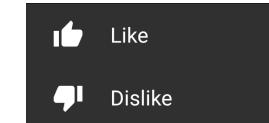
single (radio buttons)



all (check boxes)



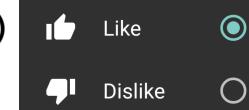
none (the default)



# How to handle clicks for "single"

- ❖ You need to programmatically set **single** checkable items to the checked state; however, others in the group will be unset automatically

```
void OnClick(object sender, Toolbar.MenuItemEventArgs e)
{
    switch (e.Item.ItemId)
    {
        case Resource.Id.like : e.Item.SetChecked(true); ... break;
        case Resource.Id.dislike: e.Item.SetChecked(true); ... break;
    }
}
```

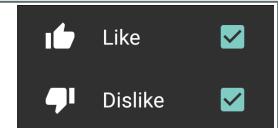


Show item state as checked

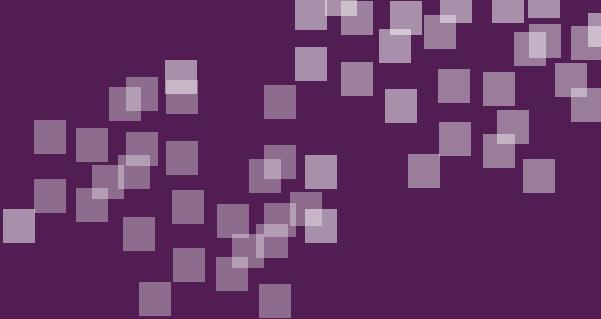
# How to handle clicks for "all"

- ❖ You need to programmatically toggle the checkable state for items in an **all**-checkable group, others in the group are **not** unset automatically

```
void OnClick(object sender, Toolbar.MenuItemEventArgs e)
{
    switch (e.Item.ItemId)
    {
        case Resource.Id.like: e.Item.SetChecked(!e.Item.IsChecked); ... break;
        case Resource.Id.dislike: e.Item.SetChecked(!e.Item.IsChecked); ... break;
    }
}
```



Toggle the checked state  
of each item when clicked



# Individual Exercise

Add a checkable submenu

# Summary

1. Define an XML menu file
2. Inflate the XML menu file into a **Toolbar**'s actions menu
3. Create hierarchical menus
4. Create checkable menu items





# Use a Toolbar as an Activity's app bar

# Tasks

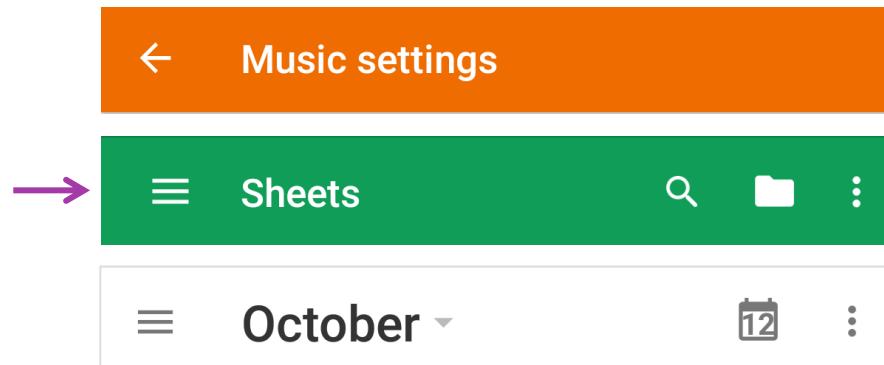
1. Set a **Toolbar** as your Activity's app bar
2. Inflate the **Toolbar** menu items
3. Respond to item click



# Motivation

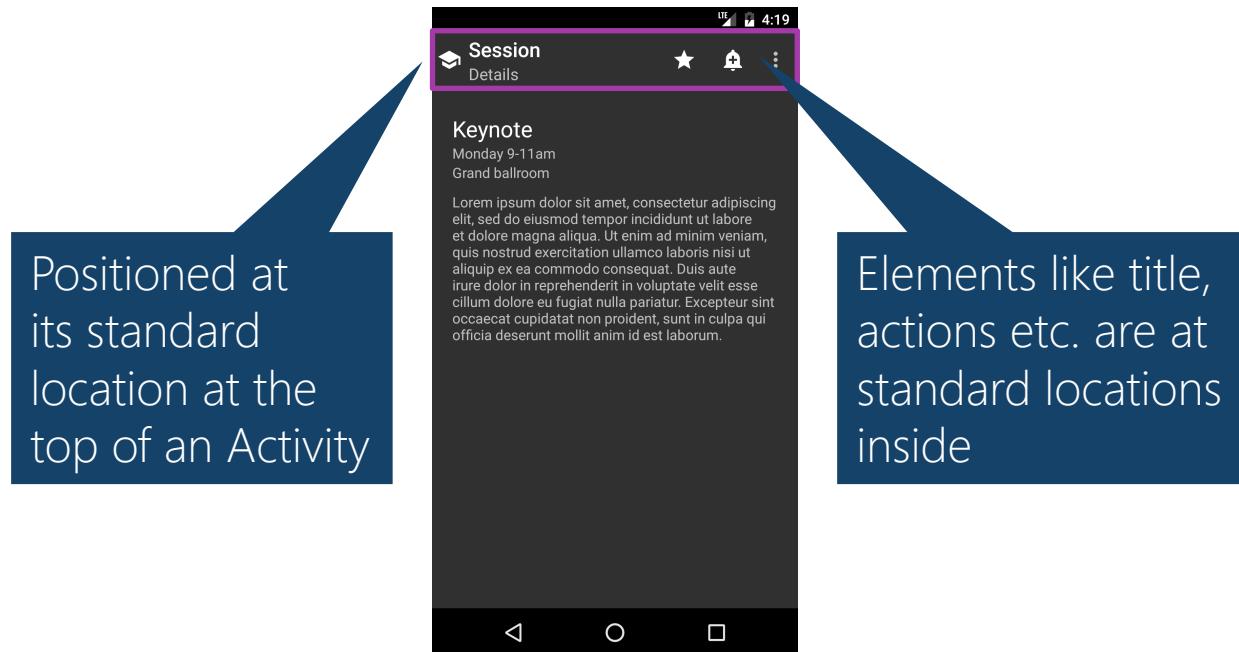
- ❖ To make your UI comfortable and familiar, you should use the standard Android structure for your Activity's navigation, title, and commands

Android users  
have seen this  
menu style in  
many apps



# What is an app bar?

- ❖ An *app bar* is a dedicated area in your UI that hosts navigation, identity, and action items



# App bar structure

- ❖ Google has guidance on how to arrange and style the elements in your app bar

Source: <https://material.google.com/layout/structure.html#structure-app-bar>

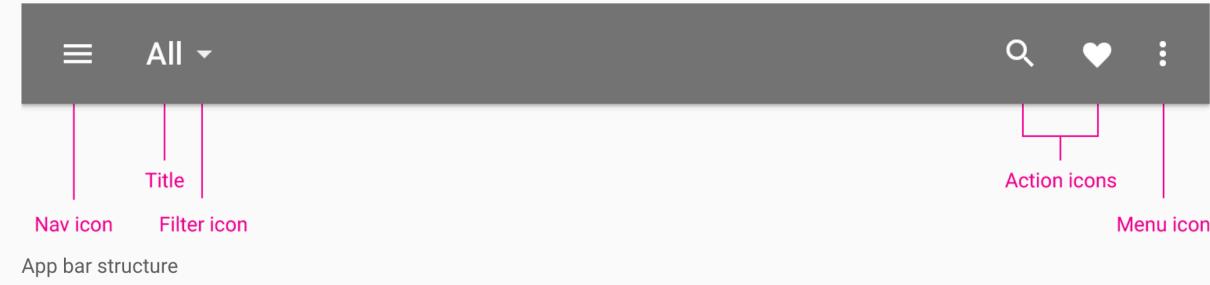
Layout – Structure

## App bar

The app bar, formerly known as the action bar in Android, is a special kind of toolbar that's used for branding, navigation, search, and actions.

The nav icon at the left side of the app bar can be:

- A control to open a navigation drawer.
- An up arrow for navigating upward through your app's



# App bar history

- ❖ *App bar* was initially called *action bar* and implemented by a class named **ActionBar**

Source: <https://developer.android.com/reference/android/support/v7/app/AppCompatActivity.html>



The header of the Android Developers website. It features a green navigation bar with the following elements: a menu icon (three horizontal lines), an Android icon, the text "Developers", and three tabs: "DESIGN", "DEVELOP", and "DISTRIBUTE". To the right is a search bar with a magnifying glass icon and the word "Search".

## getSupportActionBar

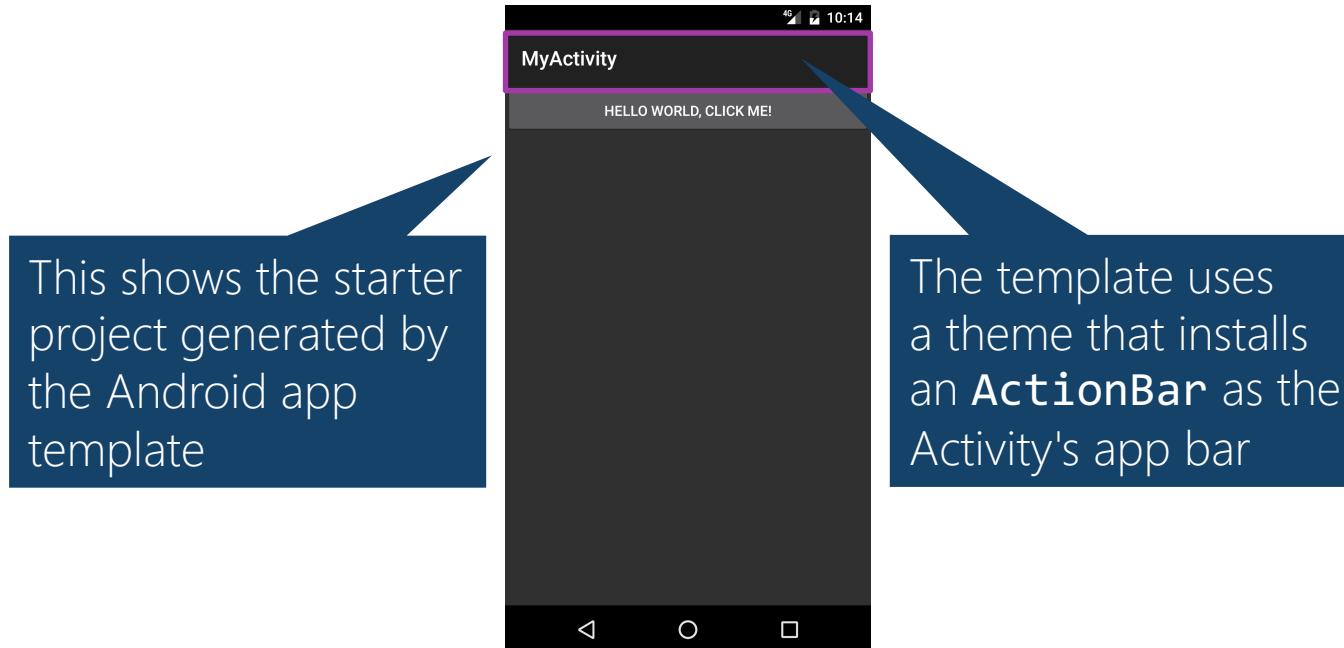
`ActionBar getSupportActionBar ()`

Support library version of `getActionBar()`.

Retrieve a reference to this activity's ActionBar.

# Default ActionBar

- ❖ By default, most app themes add an **ActionBar** to each Activity



# App bar implementation options

- ❖ You can use **ActionBar** or one of the **Toolbars** to implement your app bar; Android recommends you use the Support **Toolbar**

## ActionBar

```
public abstract class ActionBar  
extends Object  
  
java.lang.Object  
↳ android.app.ActionBar
```

Features vary based on Android version; difficult to get a uniform experience



## Toolbar

```
public class Toolbar  
extends ViewGroup  
  
java.lang.Object  
↳ android.view.View  
↳ android.view.ViewGroup  
↳ android.widget.Toolbar
```

Only available in API level 21 and higher



## Toolbar

```
public class Toolbar  
extends ViewGroup  
  
java.lang.Object  
↳ android.view.View  
↳ android.view.ViewGroup  
↳ android.support.v7.widget.Toolbar
```



Kept up to date with new features; usable with API level 9 and higher

# Activity host

- ❖ You use an Activity base class to host a Support **Toolbar** as your app bar; Android recommends you use the Support **AppCompatActivity**

Toolbar members  
only available in API  
level 21 and higher

```
public class Activity : ...  
{ ...  
    void SetSupportActionBar(Toolbar toolbar);  
    ActionBar SupportActionBar { get; }  
}
```



From the v7 support  
library; works with  
API level 9 and higher

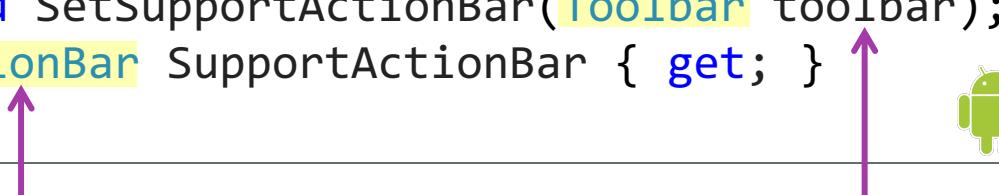
```
public class AppCompatActivity : ...  
{ ...  
    void SetSupportActionBar(Toolbar toolbar);  
    ActionBar SupportActionBar { get; }  
}
```



# Toolbar wrapper

- ❖ For compatibility with older APIs that used **ActionBar**, the **AppCompatActivity** class wraps your **Toolbar** in an **ActionBar**

```
public class AppCompatActivity : ...
{
    ...
    void SetSupportActionBar(Toolbar toolbar);
    ActionBar SupportActionBar { get; }
}
```



The getter returns an **ActionBar** wrapper around your **Toolbar**

The setter method takes a **Toolbar**

# Activity integration

- ❖ When you use a **Toolbar** as your app bar, you use Activity and **ActionBar** methods to work with it, not **Toolbar** methods

	Inflate	Set navigation icon	Item click
Standalone Toolbar	Toolbar's inflate method	XML attribute	Toolbar event
Toolbar as app bar	Override Activity method	ActionBar methods	Override Activity method

# Implement an app bar [steps]

- ❖ Several steps required to install a **Toolbar** as your Activity's app bar

1 Inherit from **AppCompatActivity**

2 Use **AppCompat.NoActionBar** theme

3 Create and position a **Toolbar**

4 Set the **Toolbar** as your app bar

5 Populate your Toolbar

6 Respond to item click

# Implement an app bar [step 1]

- ❖ Every Activity that uses a Support **Toolbar** as its app bar must inherit from **AppCompatActivity**

```
public class MainActivity : Android.Support.V7.App.AppCompatActivity  
{  
    ...  
}
```



You will need some members you inherit from this base



Inherit from AppCompatActivity

# Implement an app bar [step 2]

- ❖ **AppCompatActivity** includes an action bar by default with most themes, use one of the **NoActionBar** themes to remove it

AndroidManifest.xml

```
<application  
    ...  
    android:theme="@style/Theme.AppCompat.NoActionBar">  
</application>
```



Disable default  
action bar

# Implement an app bar [step 3]

- ❖ You will typically use XML to create and position your **Toolbar**

Position the  
**Toolbar** at  
the top of  
your UI

```
<LinearLayout ...>  
    ...  
    <android.support.v7.widget.Toolbar  
        android:id="@+id/toolbar"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content" />  
    ...  
</LinearLayout>
```

# Implement an app bar [step 4]

- ❖ During Activity creation, call **SetSupportActionBar** to install your **Toolbar** as your Activity's app bar

```
public class MainActivity : Android.Support.V7.App.AppCompatActivity
{
    ...
    protected override void OnCreate(Bundle savedInstanceState)
    {
        ...
        var toolbar = FindViewById<Android.Support.V7.Widget.Toolbar>(Resource.Id.toolbar);
        base.SetSupportActionBar(toolbar);
    }
}
```



Set **Toolbar** as your app bar

# Implement an app bar [step 5]

- ❖ Override `OnCreateOptionsMenu` to populate your `Toolbar`'s actions

```
public class MainActivity : Android.Support.V7.App.AppCompatActivity
{
    ...
    public override bool OnCreateOptionsMenu(Android.Views.IMenu menu)
    {
        base.MenuInflater.Inflate(Resource.Menu.actions, menu);
        return true;
    }
}
```

Use inherited  
inflater

Populate actions using a  
standard XML menu file

# Implement an app bar [step 6]

- ❖ Override `OnOptionsItemSelected` to respond to app bar item click

```
public class MainActivity : Android.Support.V7.App.AppCompatActivity
{
    ...
    public override bool OnOptionsItemSelected(Android.Views.IMenuItem item)
    {
        switch (item.ItemId)
        {
            ...
        }
    }
}
```

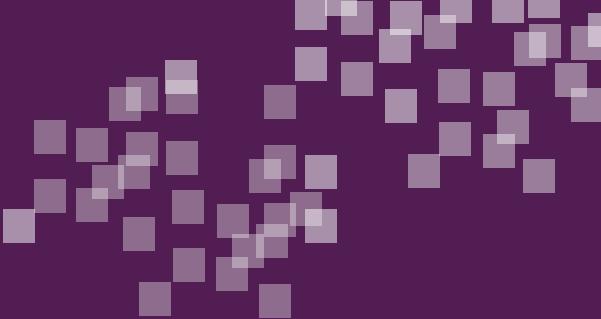


Identifies which item  
the user selected



6

Respond to item click



# Individual Exercise

Use a Toolbar as an Activity's app bar

# Summary

1. Set a **Toolbar** as your Activity's app bar
2. Inflate the **Toolbar** menu items
3. Respond to item click





# Set a navigation icon on an app bar

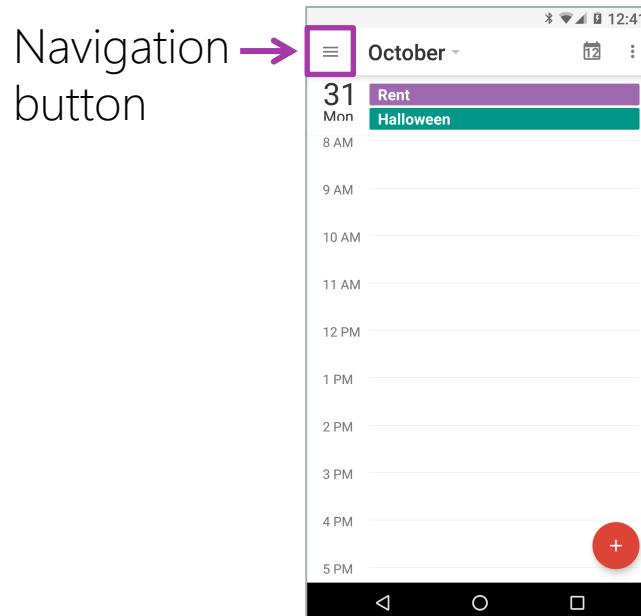
# Tasks

1. Enable the app bar navigation button
2. Set the navigation icon
3. Respond when the navigation button is clicked



# Motivation

- ❖ Some navigation patterns use a button in the top-left corner of the Activity's UI



# ActionBar support for app bar navigation

- ❖ The **ActionBar** class supplies the methods you use to set up navigation in your app bar

```
public class ActionBar : ...  
{    ...  
    void SetDisplayHomeAsUpEnabled(bool showHomeAsUp);  
    void SetHomeAsUpIndicator(int resId);  
}
```

Enable the navigation button



Select the icon you want to display on the button



The methods were originally for Up navigation and the API names remain even though they are now used for several navigation styles

# Add navigation [steps]

- ❖ Several steps required to add navigation to your app bar

- 1 Enable navigation
- 2 Set the icon
- 3 Respond to item click

# Add navigation [step 1]

- ❖ Use **SetDisplayHomeAsUpEnabled** to turn on the navigation button within your app bar

```
public class MainActivity : Android.Support.V7.App.AppCompatActivity
{
    ...
    protected override void OnCreate(Bundle savedInstanceState)
    {
        ...
        SupportActionBar.SetDisplayHomeAsUpEnabled(true);
        ...
    }
}
```



Show the navigation button

# Add navigation [step 2]

- ❖ Use **SetHomeAsUpIndicator** to specify a navigation icon

```
public class MainActivity : Android.Support.V7.App.AppCompatActivity
{
    ...
    protected override void OnCreate(Bundle savedInstanceState)
    {
        ...
        SupportActionBar.SetHomeAsUpIndicator(Resource.Drawable.ic_menu_white_24dp);
        ...
    }
}
```



Choose your navigation icon

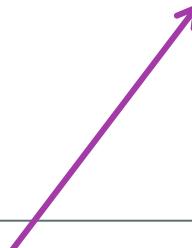


Set the icon

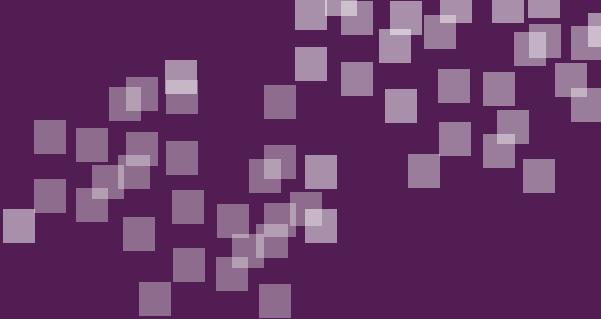
# Add navigation [step 3]

- ❖ Navigation button clicks are reported via `OnOptionsItemSelected`

```
public class MainActivity : Android.Support.V7.App.AppCompatActivity
{
    ...
    public override bool OnOptionsItemSelected(Android.Views.IMenuItem item)
    {
        if (item.ItemId == Android.Resource.Id.Home)
        {
        }
    }
}
```



The navigation button is identified by the special Android Id "**Home**"



# Individual Exercise

Set a navigation icon on an app bar

# Summary

1. Enable the app bar navigation button
2. Set the navigation icon
3. Respond when the navigation button is clicked



# Thank You!

Please complete the class survey in your profile:  
[university.xamarin.com/profile](http://university.xamarin.com/profile)

