# ITW202: Mobile Application

## Unit IV: Developing for Android

Ms. Sonam Wangmo

Gyalpozhing College of Information Technology
Royal University of Bhutan

May 5, 2021

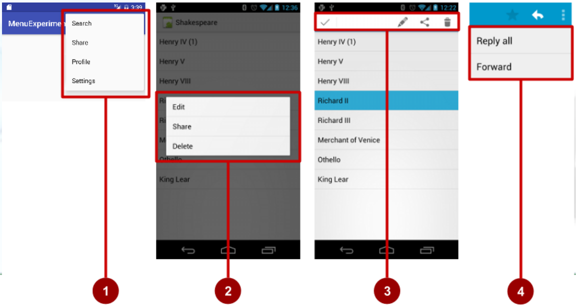
## Menus and pickers

# Contents

## Overview

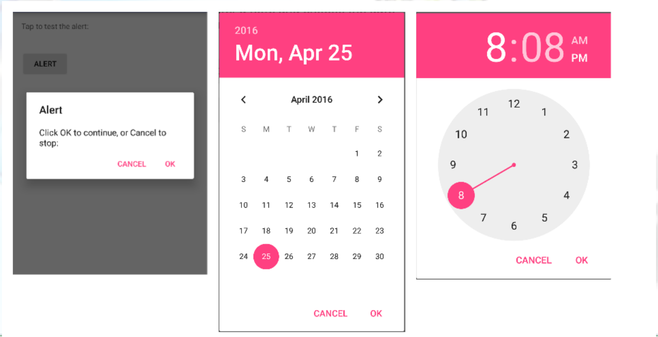
# Types of Menus

1. App bar with options menu
2. Context menu
3. Contextual action bar
4. Popup menu



# Dialogs and pickers

1. Alert dialog
2. Date picker
3. Time picker

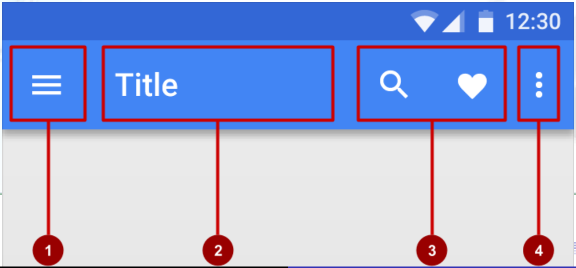


## App Bar with Options Menu

## What is the App Bar?

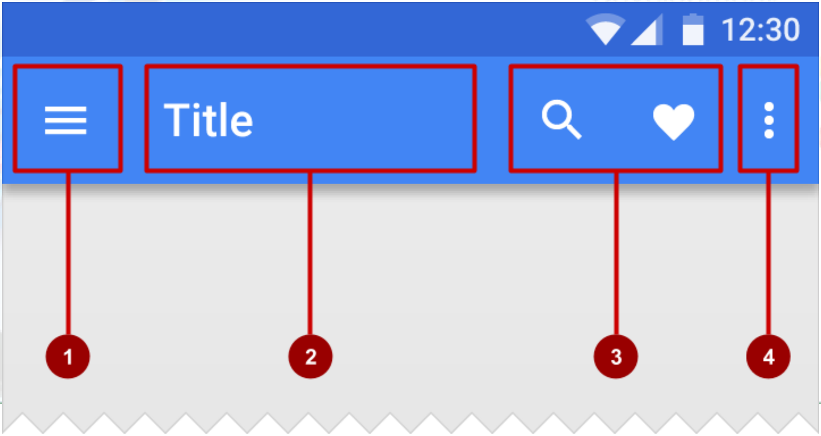
Bar at top of each screen—same for all devices (usually)

1. Nav icon to open navigation drawer
2. Title of current Activity
3. Icons for options menu items
4. Action overflow button for the rest of the options menu



# What is the App Bar?

Bar at top of each screen—same for all devices (usually)



## What is the App Bar?

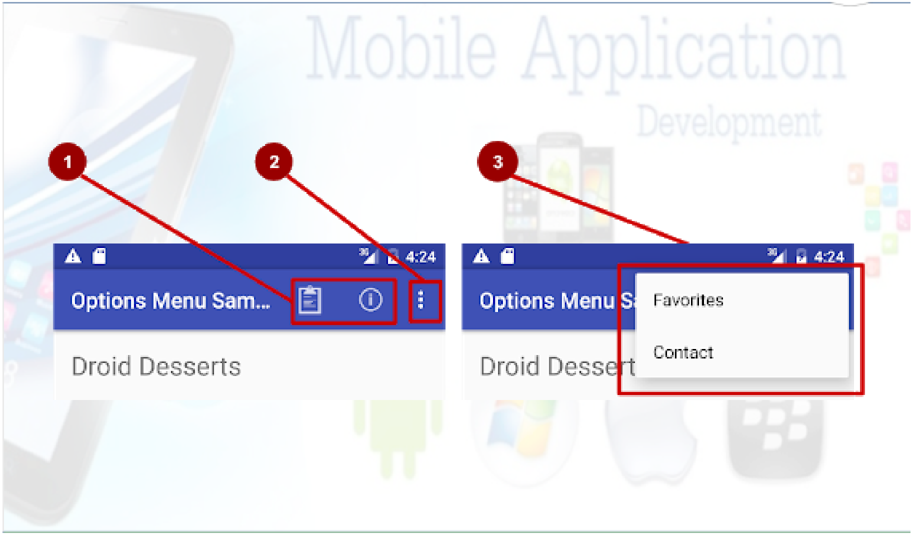
Bar at top of each screen—same for all devices (usually)

1. Navigation button or Up button: Use a navigation button in this space to open a navigation drawer, or use an Up button for navigating up through your app's screen hierarchy to the parent activity.
2. Title: The title in the app bar is the app title, or the name defined in AndroidManifest.xml by the android:label attribute for the activity.

## What is the App Bar?

3. Action icons for the options menu: Each action icon appears in the app bar and represents one of the options menu's most frequently used items. Less frequently used options menu items appear in the overflow options menu.
4. Overflow options menu: The overflow icon opens a popup with option menu items that are not shown as icons in the app bar.

# What is the options menu?



## What is the options menu?

- Action icons in the app bar for important items (1)
- Tap the three dots, the "action overflow button" to see the options menu (2)
- Appears in the right corner of the app bar (3)
- For navigating to other activities and editing app settings

## Adding the app bar

- Each activity that uses the default theme also has an ActionBar as its app bar.
- When you start an app from a template such as Empty Activity, an ActionBar appears as the app bar.

Features were added to the native ActionBar over time, so the behavior of the native ActionBar depends on the version of Android that the device is running.

## Adding the app bar

For this reason, if you add an options menu, use the v7 appcompat support library's Toolbar as an app bar:

- Toolbar makes it easy to set up an app bar that works on a wide range of devices.
- Toolbar gives you room to customize your app bar later, as your app develops.
- Toolbar includes the most recent features, and it works for any device that can use the support library.

## How to add toolbar

The following are the general steps:

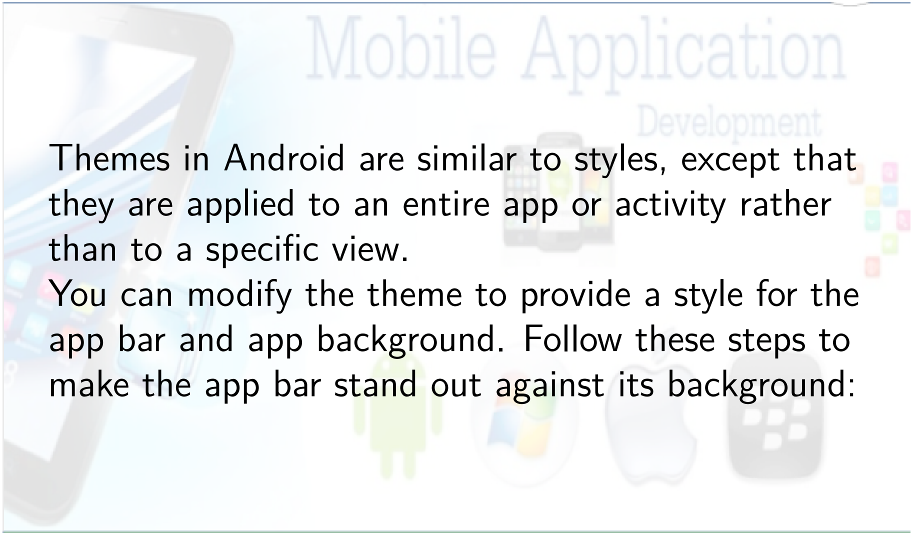
1. Add the support libraries appcompat and design.
2. Use a NoActionBar theme and styles for the app bar and background.
3. Add an AppBarLayout and a Toolbar to the layout.
4. Add code to the Activity to set up the app bar

## 1. Adding the support libraries

Open the build.gradle file for your app, and add the support library feature project identifiers to the dependencies section. For example, to include support:appcompat

- implementation 'androidx.appcompat:appcompat:1.2.0'

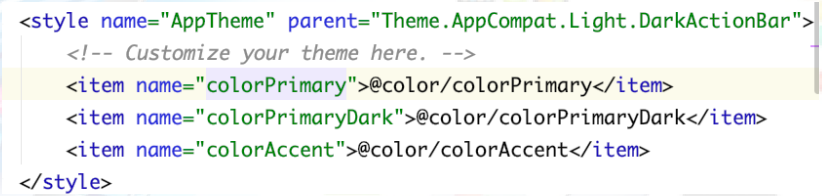
## 2. Using themes to design the app bar



Themes in Android are similar to styles, except that they are applied to an entire app or activity rather than to a specific view.
You can modify the theme to provide a style for the app bar and app background. Follow these steps to make the app bar stand out against its background:

## 2. Using themes to design the app bar

a. Open theme.xml. You should already have the following in the file within the <resources> section:



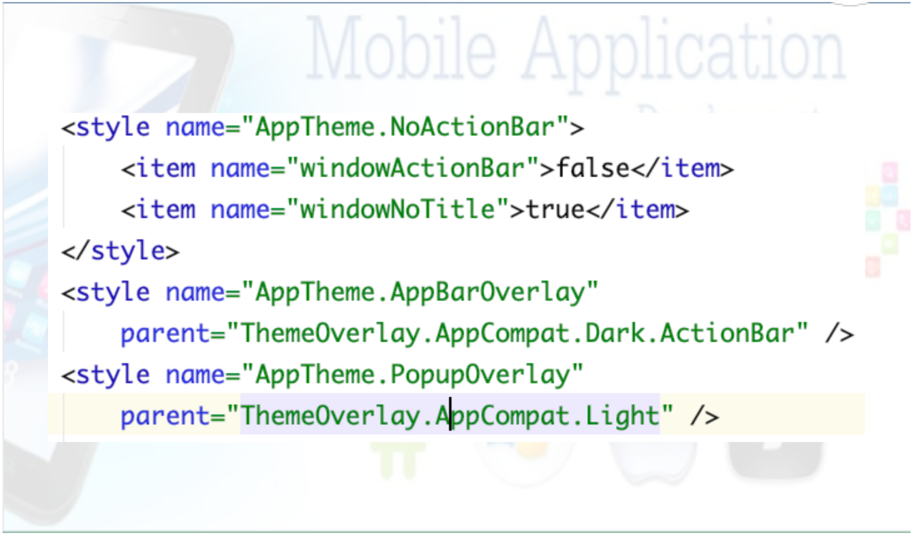
```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

Note: Theme.AppCompat.Light.DarkActionBar, which is a standard theme supplied with Android.

## 2. Using themes to design the app bar

b. Add the AppTheme.NoActionBar, AppTheme.AppBarOverlay, and AppTheme.PopupOverlay styles under the AppTheme style, as shown below

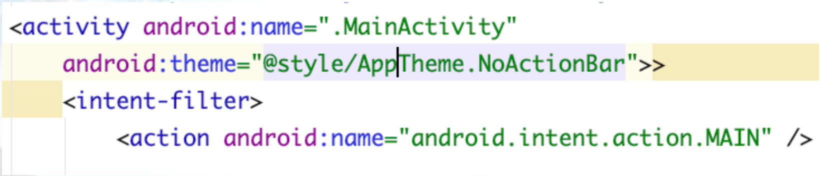
## 2. Using themes to design the app bar



```
<style name="AppTheme.NoActionBar">
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
</style>
<style name="AppTheme.AppBarOverlay"
    parent="ThemeOverlay.AppCompat.Dark.ActionBar" />
<style name="AppTheme.PopupOverlay"
    parent="ThemeOverlay.AppCompat.Light" />
```

## 2. Using themes to design the app bar

c. Open AndroidManifest.xml and add the NoActionBar theme in appcompat to the <application> element. Using this theme prevents the app from using the native ActionBar class to provide the app bar:

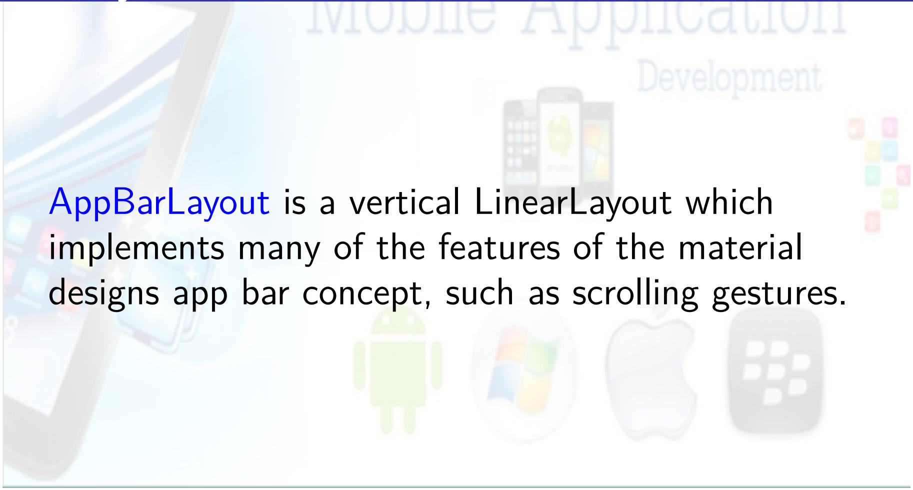


```
<activity android:name=".MainActivity"
    android:theme="@style/AppTheme.NoActionBar">>
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
```

## 3. Adding AppBarLayout and a Toolbar to the layout

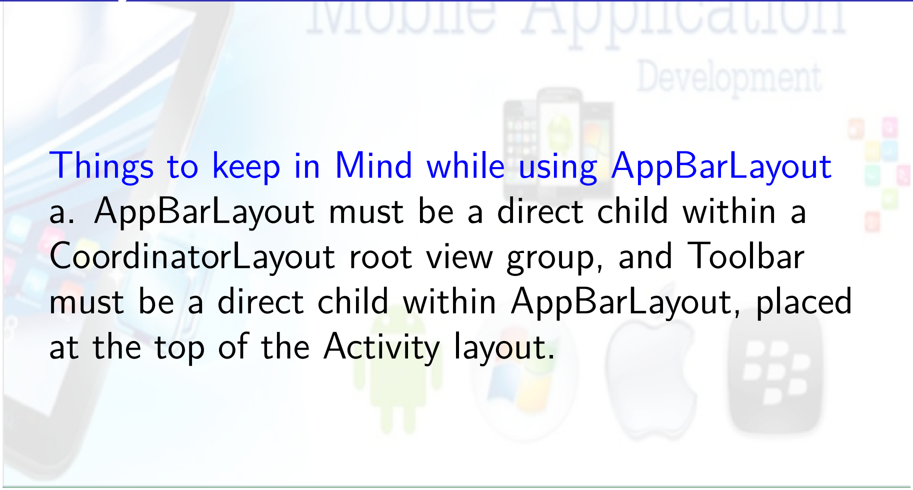
Note: If you start an app project using the Basic Activity template, the template adds the AppBarLayout and Toolbar for you, so you can skip this step.
If you are not using the Basic Activity template, you can include the Toolbar in an Activity layout by adding an AppBarLayout and a Toolbar element.

## 3. Adding AppBarLayout and a Toolbar to the layout



AppBarLayout is a vertical LinearLayout which implements many of the features of the material designs app bar concept, such as scrolling gestures.

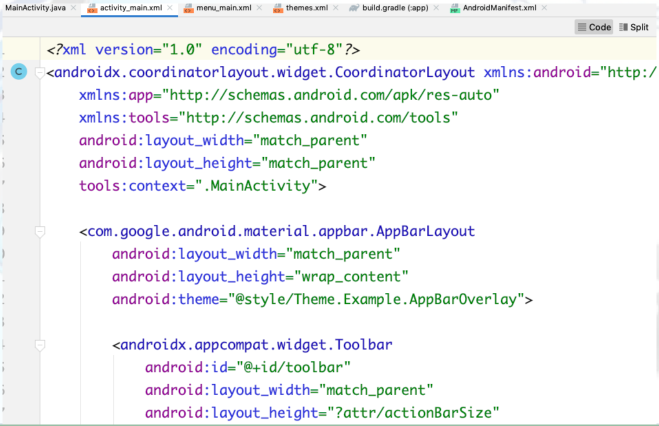
## 3. Adding AppBarLayout and a Toolbar to the layout



Things to keep in Mind while using AppBarLayout

a. AppBarLayout must be a direct child within a CoordinatorLayout root view group, and Toolbar must be a direct child within AppBarLayout, placed at the top of the Activity layout.

# 3. Adding AppBarLayout and a Toolbar to the layout



```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http:/
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/Theme.Example.AppBarOverlay">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
```

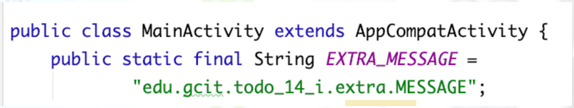
## 3. Adding AppBarLayout and a Toolbar to the layout

b. AppBarLayout also requires a separate content layout sibling for the content that scrolls underneath the app bar. You can add this sibling as a view group (such as RelativeLayout or LinearLayout) in the same layout file, or in a separate layout file.
content_main.xml

## 4. Adding code to set up the app bar

Note: If you start an app project using the Basic Activity template, the template adds the code needed to set up the app bar, so you can skip this step.

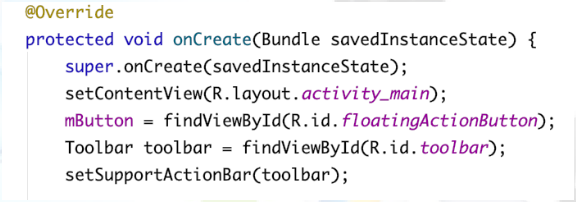
a. Make sure that any Activity that you want to show an app bar extends AppCompatActivity:



```
public class MainActivity extends AppCompatActivity {
    public static final String EXTRA_MESSAGE =
            "edu.gcit.todo_14_i.extra.MESSAGE";
```

## 4. Adding code to set up the app bar

b. In the onCreate() method for the Activity, call setSupportActionBar() with the Toolbar. The setSupportActionBar() method sets the Toolbar as the app bar for the Activity:



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mButton = findViewById(R.id.floatingActionButton);
    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
```

The Activity now shows the app bar. By default, the app bar contains just the name of the app.

## Adding the options menu

- Android provides a standard XML format to define options menu items.
- Instead of building the menu in your Activity code, you can define the menu and all its items in an XML menu resource.
- A menu resource defines an application menu (options menu, context menu, or popup menu) that can be inflated with MenuInflater, which loads the resource as a Menu object in your Activity.

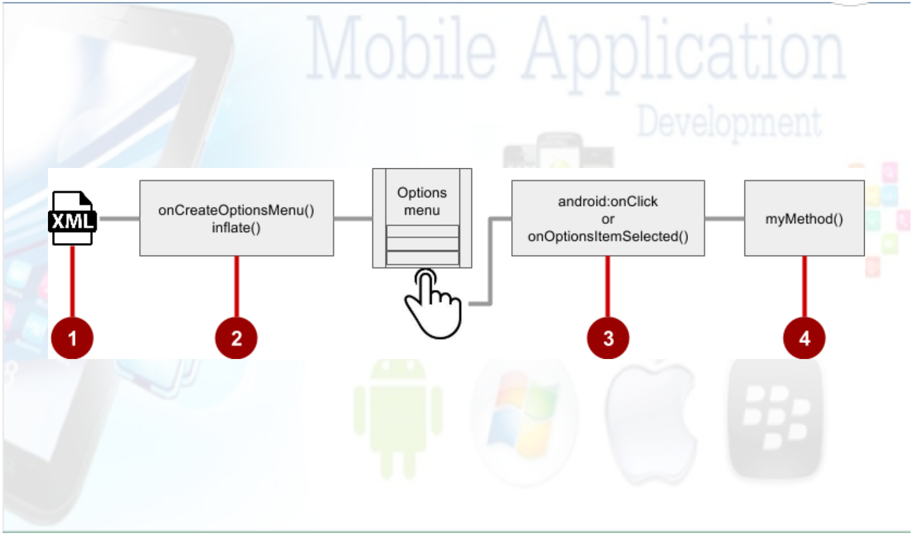
# Steps to implement options menu

1. XML menu resource. Create an XML menu resource file for the menu items, and assign appearance and position attributes as described in the next section. I
2. inflating the menu. Override the onCreateOptionsMenu() method in your Activity to inflate the menu.

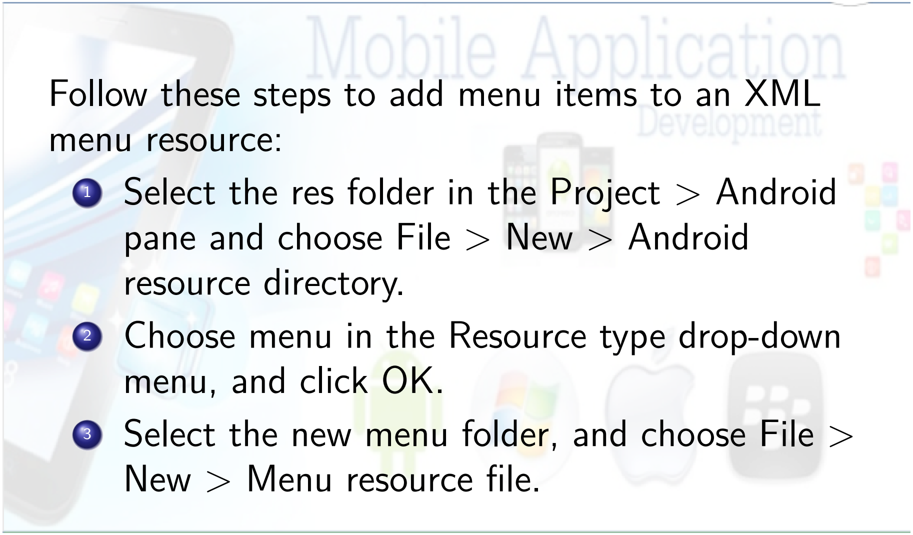
# Steps to implement options menu

3. Handling menu-item clicks. Menu items are View elements, so you can use the android:onClick attribute for each menu item. However, the onOptionsItemSelected() method can handle all the menu-item clicks in one place and determine which menu item the user clicked, which makes your code easier to understand.
4. Performing actions. Create a method to perform an action for each options menu item.

## Steps to implement options menu



## Creating an XML resource for the menu



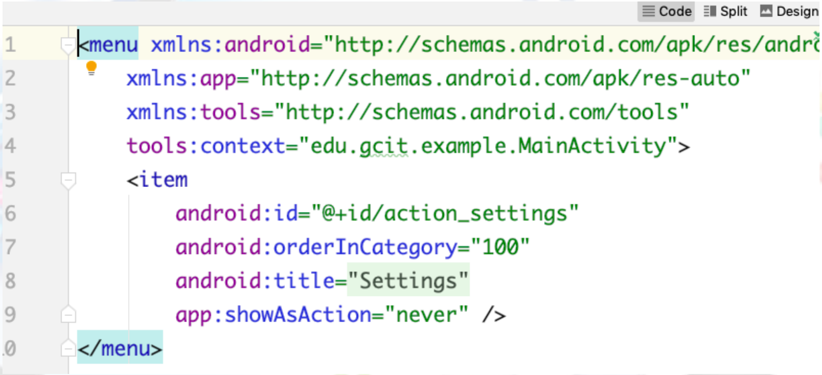
Follow these steps to add menu items to an XML menu resource:

1. Select the res folder in the Project > Android pane and choose File > New > Android resource directory.
2. Choose menu in the Resource type drop-down menu, and click OK.
3. Select the new menu folder, and choose File > New > Menu resource file.

## Creating an XML resource for the menu

4. Enter the name, such as menu_main, and click OK. The new menu_main.xml file now resides within the menu folder.
5. Open menu_main.xml and click the Text tab to show the XML code.
6. Add menu items using the <item ... /> tag. In this example, the item is Settings

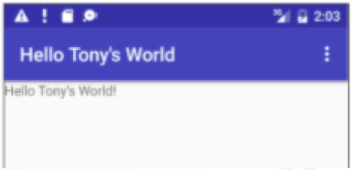
# Creating an XML resource for the menu

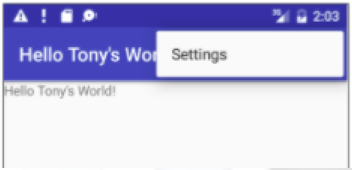


```
<menu xmlns:android="http://schemas.android.com/apk/res/andr
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="edu.gcit.example.MainActivity">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="Settings"
        app:showAsAction="never" />
</menu>
```

# Creating an XML resource for the menu

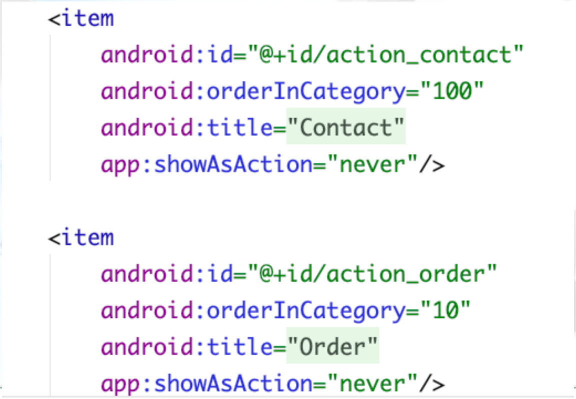
After setting up and inflating the XML resource in the Activity, the overflow icon in the app bar, when clicked, would show the options menu with just one option (Settings).





## Defining how menu items appear

To add more options menu items, add more <item ... /> tags in the menu_main.xml file.



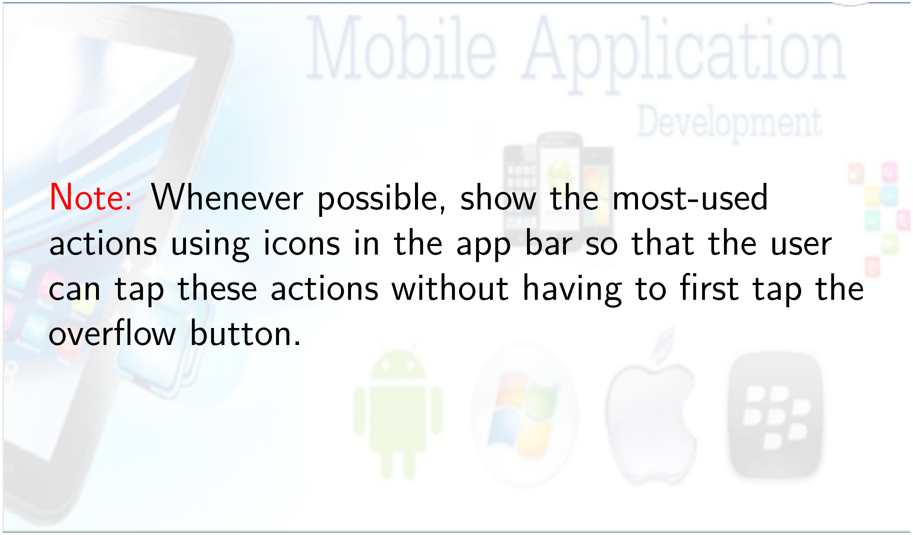
```
<item
    android:id="@+id/action_contact"
    android:orderInCategory="100"
    android:title="Contact"
    app:showAsAction="never"/>

<item
    android:id="@+id/action_order"
    android:orderInCategory="10"
    android:title="Order"
    app:showAsAction="never"/>
```

## Defining how menu items appear

Within each <item ... /> tag, you add attributes that define how the menu item appears.
The following items are placed in the overflow menu:

- Any item that you set to not appear in the app bar.
- Any item that can't fit in the app bar, given the display orientation.

# Defining how menu items appear



Note: Whenever possible, show the most-used actions using icons in the app bar so that the user can tap these actions without having to first tap the overflow button.

## Adding icons for menu items

To specify icons for actions, first add the icons as image assets to the drawable folder by following the steps below.

1. Expand res in the Project > Android pane, and right-click (or Command-click) drawable.
2. Choose New > Image Asset. The Configure Image Asset dialog appears.
3. Choose Action Bar and Tab Items in the drop-down menu.

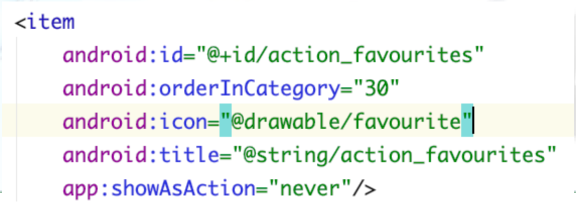
# Adding icons for menu items

4. Edit the name of the icon (for example, ic_order_white for the Order menu item).
5. Click the clip art image (the Android logo) to select a clip art image as the icon. A page of icons appears. Click the icon you want to use.
6. (Optional) Choose HOLO_DARK from the Theme drop-down menu. This sets the icon to be white against a dark-colored (or black) background. Click Next.
7. Click Finish in the Confirm Icon Path dialog.

## Icon and appearance attributes

Use the following attributes to govern the menu item's appearance:

- android:icon: An image to use as the menu item icon.
- android:title: A string for the title of the menu item.

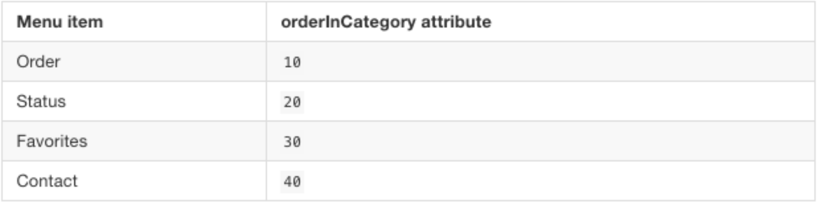


```
<item
    android:id="@+id/action_favourites"
    android:orderInCategory="30"
    android:icon="@drawable/favourite"
    android:title="@string/action_favourites"
    app:showAsAction="never"/>
```

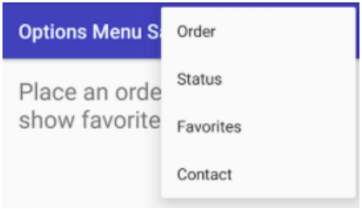
## Icon and appearance attributes

- android:orderInCategory : attribute to specify the order in which the menu items appear in the menu, with the lowest number appearing higher in the menu. This is usually the order of importance of the item within the menu.

# Icon and appearance attributes



| Menu item | orderInCategory attribute |
|---|---|
| Order | `10` |
| Status | `20` |
| Favorites | `30` |
| Contact | `40` |



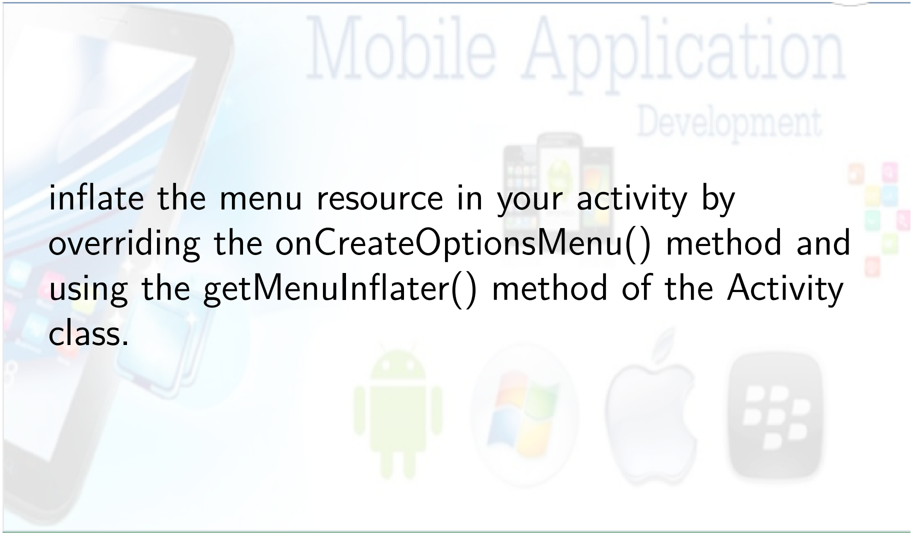
Place an orde
show favorite

## Icon and appearance attributes

Use the app:showAsAction attribute to show menu items as icons in the app bar, with the following values:

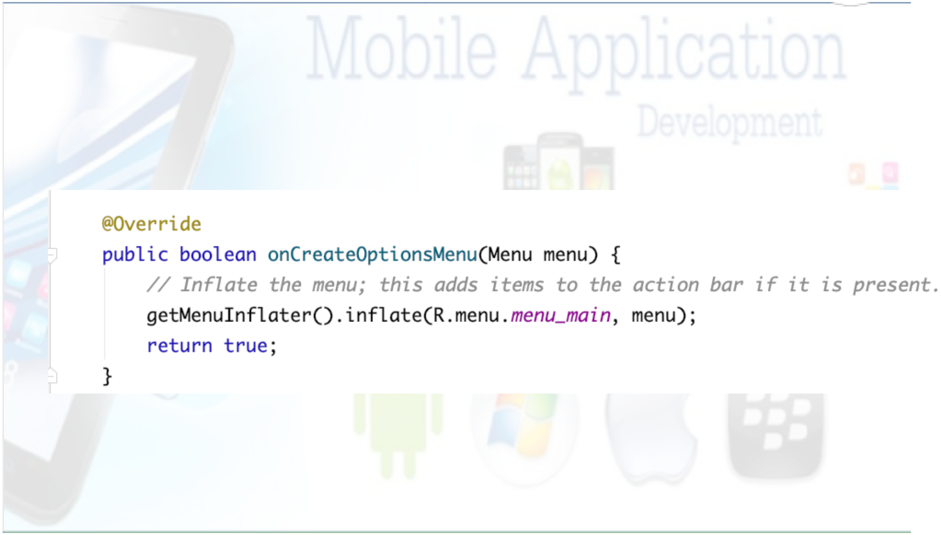
- "always": Always place this item in the app bar. Use this only if it's critical that the item appear in the app bar (such as a Search icon).
- "ifRoom": Only place this item in the app bar if there is room for it.
- "never": Never place this item in the app bar.
- "withText": Also include the title text (defined by android:title) with the item.

## Inflating the menu resource



inflate the menu resource in your activity by overriding the onCreateOptionsMenu() method and using the getMenuInflater() method of the Activity class.

# Inflating the menu resource

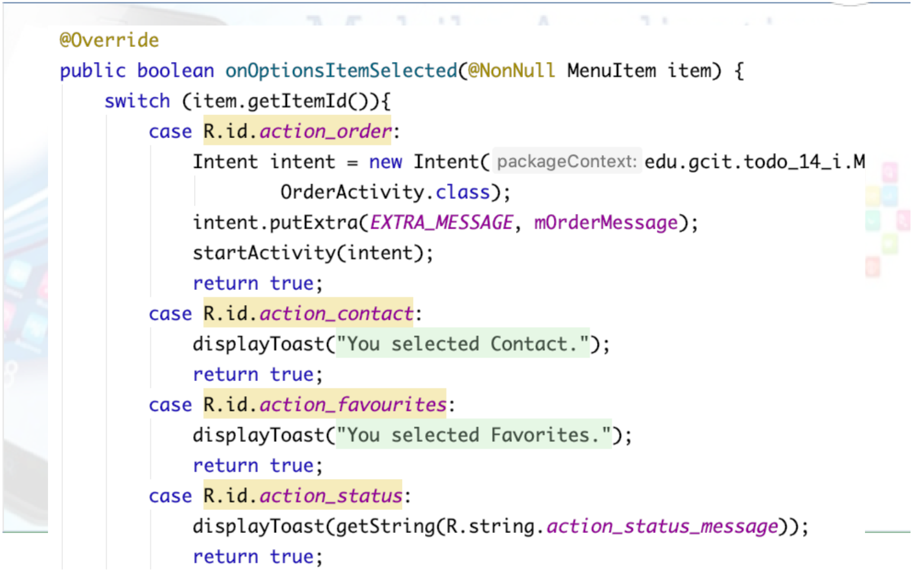


```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
```

## Handling the menu-item click

The onOptionsItemSelected() method can handle all the menu-item clicks in one place and determine which menu item the user clicked. This makes your code easier to understand.

# Handling the menu-item click



```
@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    switch (item.getItemId()){
        case R.id.action_order:
            Intent intent = new Intent( packageContext: edu.gcit.todo_14_i.M
                    OrderActivity.class);
            intent.putExtra(EXTRA_MESSAGE, mOrderMessage);
            startActivity(intent);
            return true;
        case R.id.action_contact:
            displayToast("You selected Contact.");
            return true;
        case R.id.action_favourites:
            displayToast("You selected Favorites.");
            return true;
        case R.id.action_status:
            displayToast(getString(R.string.action_status_message));
            return true;
```

THANK YOU