Advanced Mobile Application Development
Week 13: Android App bar

App bar

The App Bar is a consistent navigation element that is standard throughout modern Android applications. https://material.io/design/components/app-bars-top.html#anatomy
The App Bar can consist of:

- Consistent navigation (including navigation drawer)
- An application icon
- An "upward" navigation to logical parent
- An application or activity-specific title
- Primary action icons for an activity
- Overflow menu

Terminology
The app bar is often called the action bar, but there is a difference. App bar is the general name, it can be implemented as an action bar or a toolbar.
Beginning with API level 11 all activities that use the default theme have an ActionBar as an app bar. However, app bar features have gradually been added to the native ActionBar over various Android releases. As a result, the native ActionBar behaves differently depending on what version of the Android system a device may be using.
The most recent features are added to the support library's version of Toolbar, and they are available on any device that can use the support library. Because of this you should use the support library's Toolbar class to implement your activities' app bars. Using the support library's toolbar helps ensure that your app will have consistent behavior across the widest range of devices.

Menus

Android has three different types of menus:

1. Options menu: these are actions that are accessed in the app bar
2. Context menu: a floating menu that is often presented on a long click event. It provides actions that affect the selected content
3. Popup menu: a menu that pops up with a vertical list of items. These are used for actions that relate to regions of content in your activity.

We'll look at how to add an options menu to our app bar.

Navigation

When your app is launched, a new activity is created and becomes the base destination of the app's back stack.
The top of the stack is the current view, and the previous destinations in the stack represent the history of where you've been. The back stack always has the start destination of the app at the bottom of the stack.
Operations that change the back stack always operate on the top of the stack, either by pushing a new destination onto the top of the stack or popping the top-most destination off the stack. Navigating to a destination pushes that destination on top of the stack.
The Navigation component manages all of your back stack ordering for you, though you can also choose to manage the back stack yourself.

Back
The Back button is in the system navigation bar at the bottom of the screen and is used to navigate in reverse-chronological order through the history of screens the user has recently worked with. When you press the Back button, the current destination is popped off the top of the back stack, and you then navigate to the previous destination.

Up
Up navigation is a way for the user to navigate to the logical parent screen in the app's hierarchy. This differs from back navigation because it is not based on the history, or the user's path, but a defined parent activity. Up navigation will therefore never exit the app.

Coffee (cont.)
The activities we've created so far are passive activities that provide information and navigation. An active activity lets the user do or create something. Active activities are usually added to the app bar to help users quickly access those activities.

Toolbar
Let's add an app bar to our Coffee app where we can access a (fake) coffee order form.

Make sure you have the androidx support library in your Gradle file as a dependency.
```
implementation 'androidx.appcompat:appcompat:1.1.0'
```

In the AndroidManifest.xml file (app/manifests) the **android:theme** attribute sets the theme. The @style prefix means that the theme is defined in a style resource file.
Open styles.xml (app/res/values)
You will probably have the following:
```
<style name="AppTheme"
parent="android:Theme.AppCompat.Light.DarkActionBar">
```
Change this or create a new style with a parent that has no action bar.
```
<style name="NoActionBar" parent="Theme.AppCompat.Light.NoActionBar">
```
You can use the colors already defined or go into colors.xml and update or add new colors.
Back in the AndroidManifest.xml file change the theme to use your new theme.
```
android:theme="@style/NoActionBar">
```
Now when you run your app you should not see an action bar.

For any activity that you want to have a toolbar you need the do the following. We'll do it for our main activity, just repeat these steps.
Go into your activity_main layout file and in the design view drag out a toolbar and add it to the top of the layout. Adjust the other views and constraints accordingly.
For the toolbar I set top, start, and end constraints equal to 0 and made the layout_width match_constraints and layout_height wrap_content.
Make sure your toolbar has an ID.

Go into MainActivity.java and in onCreate() get access to the toolbar and then set the action bar to be the toolbar.
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
    //get toolbar and set it as the app bar
    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
…
```

You'll get a message that there are multiple choices for the class to import. Choose the import for the toolbar class `import androidx.appcompat.widget.Toolbar;`

Now if you run it you should see the toolbar with the name of your app.

Now let's add buttons for user actions in the toolbar.
To add action items to the toolbar you do three things:

1. Define the action items in a menu resource file
2. Get the activity to inflate the menu resource
3. Write the code to respond to a user clicking on the item

App bar actions
All action buttons and other items available in the action overflow are defined in an XML menu.To add actions to the action bar we need a menu folder in resources.
Right click on the res folder, chose new Directory and name it menu.
You should have a menu folder nested in the res folder.
Right click on the menu folder, chose new Menu resource file
File name: menu_main
Resource type: Menu
Source set: main
Directory name: menu

I want to use an icon for my bulb order form. I used the Material Design store icon and copied it into the drawables folder. https://material.io/tools/icons
I'll also need a title for that and for Settings, which should always be in the overflow.
In strings.xml add

```
<string name="create_order">Order</string>
<string name="action_settings">Settings</string>
```

Go into menu_main.xml and add a menu item for each item you want in the toolbar.

```
<menu xmlns:android= "http://schemas.android.com/apk/res/android">
<item
    android:id="@+id/create_order"
    android:title="@string/create_order"
    android:icon="@drawable/ic_store"
    app:showAsAction="ifRoom" />

<!-- Settings, should always be in the overflow -->
<item android:id="@+id/action_settings"
    android:title="@string/action_settings"
    app:showAsAction="never" />
</menu>
```

The **android:icon** property assigns an icon for an item that will be shown if there's room. If there's no icon it will show the title.
The **app:showAsAction** property defines how you want the item to appear – always in the main action bar or only if there's room. If there's not room it will go in the overflow area(displayed in three dots). Never will mean you'll always get the 3 dots and the item will be in there.
If you have multiple items in your menu the **android:orderInCategory** property would define the order in which they appear.
You can also define different menus for different activities if you want the items to be different based on the activity the user is in.

Now that we've added action items to the menu resource file, we need to add the items to the toolbar in every activity's onCreateOptionsMenu()method where we want the menu to be accessible. It runs when the action bar's menu gets created and takes one parameter, a Menu object representing the action bar.
In MainActivity.java implement onCreateOptionsMenu() If you just start typing the name of the method you'll be able to use autocomplete and it will also add the needed import statement.

```
import android.view.Menu;

public boolean onCreateOptionsMenu(Menu menu){
    //inflate menu to add items to the action bar
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return super.onCreateOptionsMenu(menu);
}
```

Now when you run the app you should see the store icon in the action bar to indicate there's a menu and Order should be in the menu.
But you won't see this in the other views. You need to add the same method in all activities you want the menu to be a part of. You can also create the toolbar in its own layout and then include it in your activities.

```
<include
       layout="@layout/toolbar_main"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"/>
```

For your activity to respond when an action item in the action bar is clicked you need to implement the `onOptionsItemSelected()` method which is called when an item in the action bar is clicked. This will also import the MenuItem class.

```
import android.view.MenuItem;

public boolean onOptionsItemSelected(MenuItem item) {
//get the ID of the item on the action bar that was clicked
    switch (item.getItemId()){
        case R.id.create_order:
            Intent intent = new Intent(this, OrderActivity.class);
    startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
```

```
    }
}
```

The `onOptionsItemSelected()` method takes one attribute, a `MenuItem` object that represents the item on the action bar that was clicked.

You will see an error on OrderActivity until we create that class.

Order Activity
Create a new activity for the fake order form.
File | New |Activity
Either Gallery or Empty Activity
Activity name: OrderActivity
Check Generate Layout File
Layout Name: activity_order
Do not check Launcher Activity as this is not the launcher activity for our app.
Source language: Java
This creates a new layout xml file and java file for our new activity.

Once the project builds the error in MainActivity.java should be gone.

We won't be creating the order form, the main goal here was to understand how menus and menu items work.

You'll need to implement `onCreateOptionsMenu()` and `onOptionsItemSelected()` in every activity you want the menu to be added to.

Up Navigation
The app bar has an Up button that is used to navigate up the activity hierarchy. This is not the same as the back button which allows users to navigate back through the history of activities they've been to. The up button will take the user to that activity's parent activity. You define the parent activity in the AndroidManifest.xml file. We're going to make the parent of OrderActivity be MainActivity.
In the AndroidManifest.xml file add a parent for the Order activity

```
<activity android:name=".OrderActivity"
    android:parentActivityName=".MainActivity" >
</activity>
```

Add a toolbar to the activity_order layout so we can enable the Up button in the app bar.
In OrderActivity.java add to the onCreate() method.

```
//get toolbar and set it as the app bar
Toolbar toolbar = findViewById(R.id.toolbar2);
setSupportActionBar(toolbar);

//enable the up button
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
```

Now when you run the app and go to the Order activity you'll see a back arrow in the app bar which should take you back to the main activity.
If you enable the up button and the activity doesn't have a parent defined the app will crash.