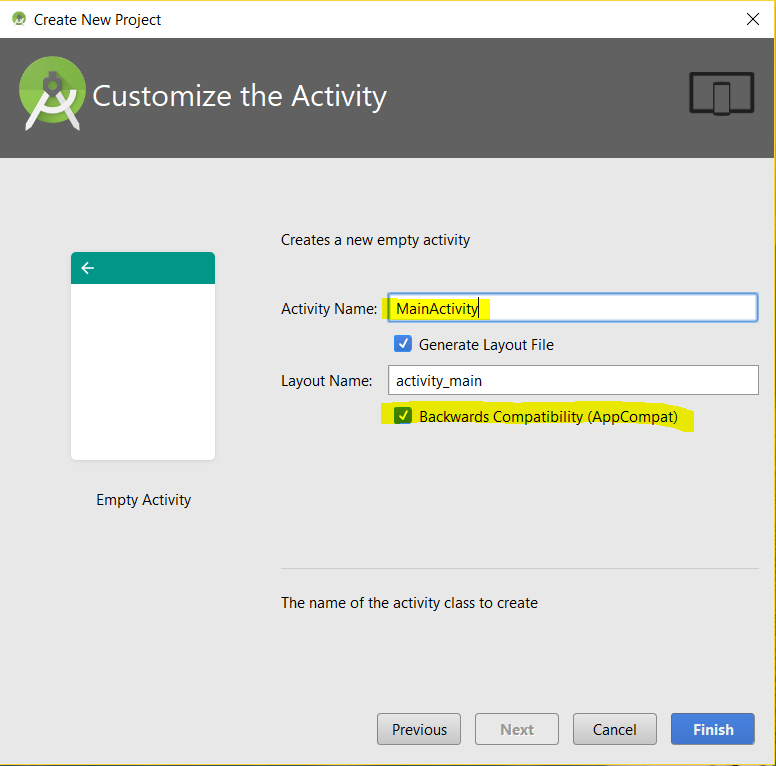
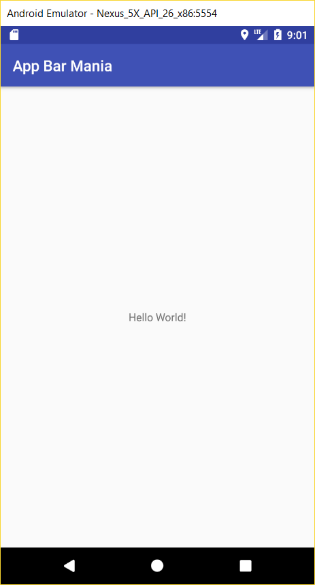
App bar and toolbars

Create a new app and name it “*App Bar Mania*”. Add an empty activity named *MainActivity* and do not uncheck “*Backwards Compatibility (AppCompat)*”.



Note that *MainActivity* inherits from *AppCompatActivity*, which is in the *v7 AppCompay library*.

package ca.bcit.appbarmania;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

}

Run the application:

Theme and colors are specified in the styles.xml and colors.xml files.

### styles.xml

<resources>

<!-- Base application theme. -->

<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>

</resources>

### colors.xml

<?xml version="1.0" encoding="utf-8"?>

<resources>

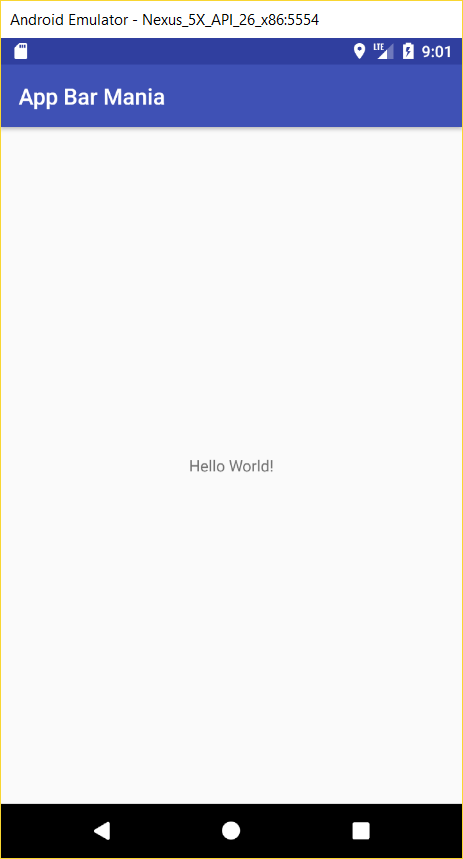
<color name="colorPrimary">#3F51B5</color>

<color name="colorPrimaryDark">#303F9F</color>

colorPrimaryDark

<color name="colorAccent">#FF4081</color>

</resources>



01. colors.xml.txt

colorPrimary

Change colors.xml to have other colors for *colorPrimary* & *colorPrimaryDark* as shown below:

<?xml version="1.0" encoding="utf-8"?>

<resources>

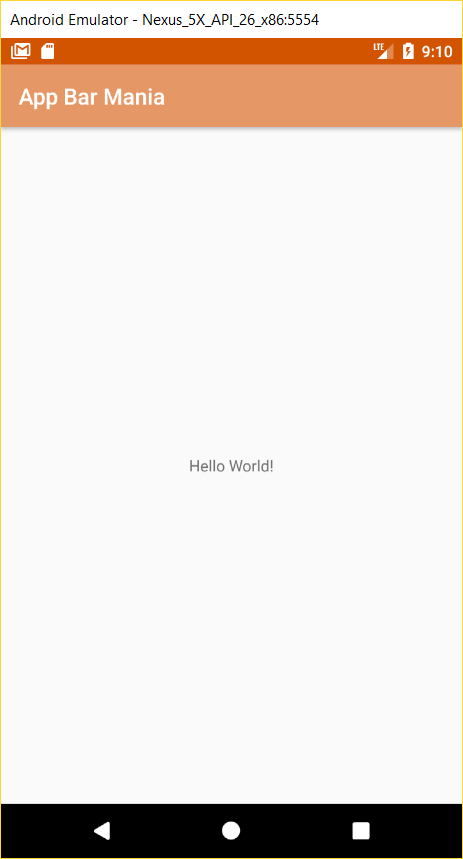
<color name="colorPrimary">#E59866</color>

<color name="colorPrimaryDark">#D35400</color>

<color name="colorAccent">#FF4081</color>

</resources>

Run the app and it should have different colors:



## How to add a toolbar:

1. Add the v7 *AppCompat Support Library* as a dependency. This is necessary because the *Toolbar* class lives in this library.
2. Make sure your activity extends the *AppCompatActivity* class.
3. Remove the existing app bar. You do this by changing the theme to one that does not include an app bar.
4. Add a toolbar to the layout
5. Update the activity and set the toolbar as the activity’s app bar.

Steps 1 & 2 are already fulfilled.

### Step 3: Remove existing app bar

In the *styles.xml* file, change *DarkActionBar* to *NoActionBar*:

<resources>

<!-- Base application theme. -->

<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">

<!-- Customize your theme here. -->

<item name="colorPrimary">@color/colorPrimary</item>

<item name="colorPrimaryDark">@color/colorPrimaryDark</item>

<item name="colorAccent">@color/colorAccent</item>

</style>

</resources>

### Step 4: Add toolbar to the layout

Add the following to *activity\_main.xml* just before the *TextView*:

02. toolbar.txt

<android.support.v7.widget.Toolbar

android:id="@+id/toolbar"

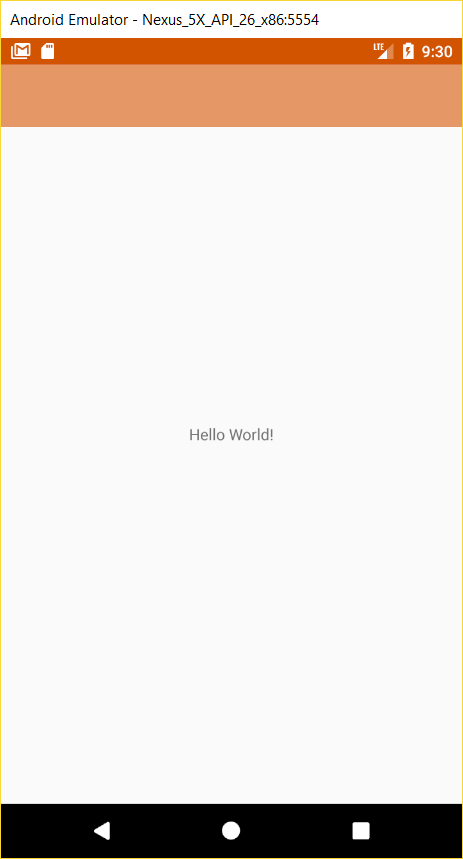
android:layout\_width="match\_parent"

android:layout\_height="?attr/actionBarSize"

android:background="?attr/colorPrimary"

android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" />

Run the app and you should see the *Toolbar*:



NOTICE THERE IS NO APP TITLE

Toolbar

It is OK to add the toolbar to the layout of a single activity app. However, this solution is not practical if you have an app that has numerous activities. The solution is to define the toolbar in a separate layout and import it into all the layouts that need it.

Create a layout named *toolbar\_main.xml* and add to it the following XML:

03. toolbar\_main.xml.txt

<?xml version="1.0" encoding="utf-8"?>

<android.support.v7.widget.Toolbar

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="?attr/actionBarSize"

android:background="?attr/colorPrimary"

android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" />

Back in *activity\_main.xml*, delete the toolbar and replace it with the following *<include..>* element:

04. include.txt

<include layout="@layout/toolbar\_main" android:id="@+id/toolbar" />

Run the app. It should behave just as the last time we run it.

### Step 5: Set toolbar as activity app bar

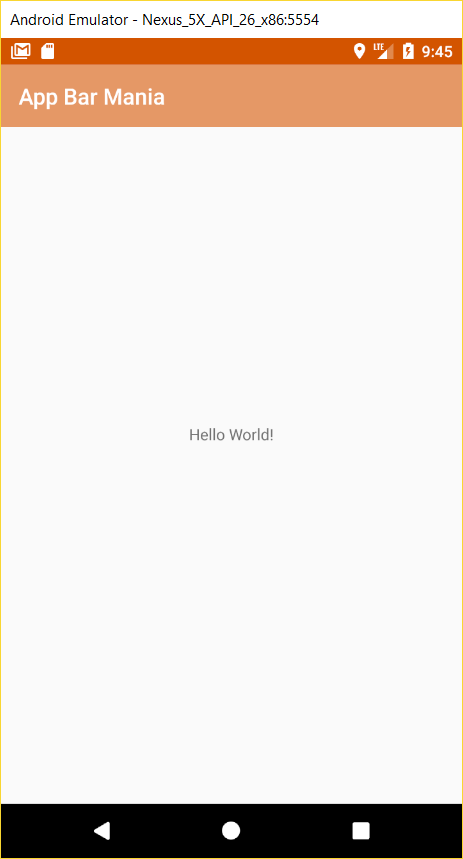
05. onCreate\_in\_MainActivity.txt

Add the following code to the bottom of the *onCreate()* method in *MainActivity*:

Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);

setSupportActionBar(toolbar);

Run the app. You should see the application title displayed in the toolbar:



App title reappeared

The next thing we will do is add actions to the app bar. These are buttons or text in the app bar that you click on to make something happen. We are going to add a “Info Request” button to the app bar. When clicked, it will start an activity called *InfoRequestActivity*.

Add an empty activity named *InfoRequestActivity* and do not uncheck “*Backwards Compatibility (AppCompat)*” checkbox.

Update *activity\_info\_request.xml* by adding the following *<include…>* element, just as we did with *activity\_main.xml*:

<include layout="@layout/toolbar\_main" android:id="@+id/toolbar" />

Update the *onCreate()* method in *InfoRequestActivity* with the following two lines of code, also just like we did with *MainActivity*:

Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);

setSupportActionBar(toolbar);

Add a string resource for the new activity’s title:

06. info\_request.txt

<string name="info\_request">Request for information</string>

We will change the app bar text when the *InfoRequestActivity* is loaded. Open *AndroidManifest.xml* and add the following attribute to the *<activity..>* elements that pertains to *InfoRequestActivity*:

android:label="@string/info\_request"

The *<activity…>* element for *InfoRequestActivity* will look like this:

07. android\_label.txt

<activity

android:name=".InfoRequestActivity"

android:label="@string/info\_request" >

</activity>

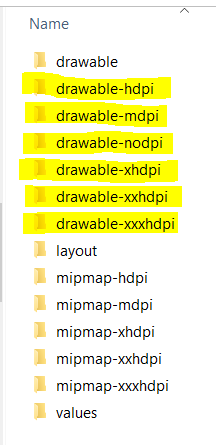
To add an action to the action bar you need to do the following:

1. Add resources for the action’s icon and text.
2. Define the action in a menu resource
3. Get the activity to add the menu resource to the app bar
4. Add code to say what the actions should do when clicked.

## Step 1: Add resources for the action’s icon and text.

Use the icons that are in the snippets *icons* directory and add them to the *res* directory in your project. Your *res* directory structure should look like this:

You can get more icons from https://material.io/icons.



Create a title for the action in *strings.xml* as follows:

08. info\_request\_title.txt

<string name="info\_request\_title">Info Request</string>

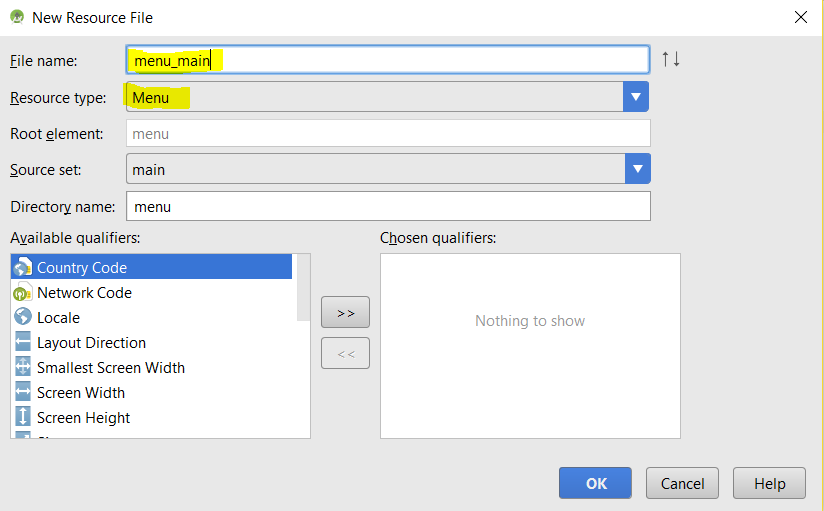
### Step 2: Define the action in a menu resource

A menu resource file tells Android what actions you want to appear on the app bar.

Right-click on the res node in Android Studio >> New >> Android resource file

File name: *menu\_main*

Resource type: *Menu*



… then click on OK.

09. menu\_main.xml.txt

Update your *menu\_main.xml* so it looks like this:

<?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">

<item android:id="@+id/action\_request\_info"

android:title="@string/info\_request\_title"

android:icon="@drawable/ic\_add\_white\_24dp"

android:orderInCategory="1"

app:showAsAction="ifRoom" />

</menu>

The *showAsAction* attribute can take the following values:

|  |  |
| --- | --- |
| ifRoom | Place the item in the app bar if there is space. Otherwise, put in the overflow. |
| withText | Include the item’s title text. |
| never | Put the item in the overflow area, and never in the main app bar. |
| always | Always place the item in the main area of the app bar. Use this option sparingly; if you apply this to many items they may overlap each other. |

### Step 3: Get the activity to add the menu resource to the app bar

Implement the *onCreateOptionsMenu()* method in *MainActivity*. Add the following method to *MainActivity*:

10. onCreateOptionsMenu.txt

@Override

public boolean onCreateOptionsMenu(Menu menu) {

// Inflate the menu. This adds items to the app bar.

getMenuInflater().inflate(R.menu.menu\_main, menu);

return super.onCreateOptionsMenu(menu);

}

To make your app react when an action in the app bar is clicked, you implement the *onOptionsItemSelected()* method in *MainActivity* with the following code:

11. onOptionsItemSelected.txt

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.action\_request\_info:

Intent i = new Intent(this, InfoRequestActivity.class);

startActivity(i);

return true;

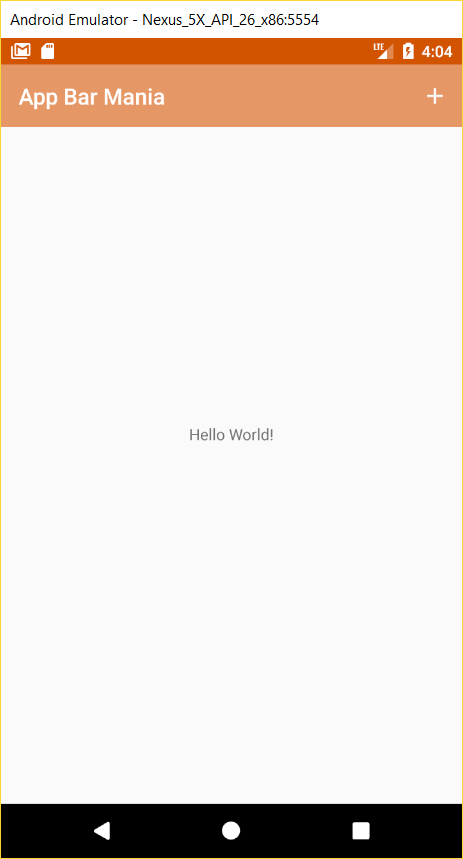
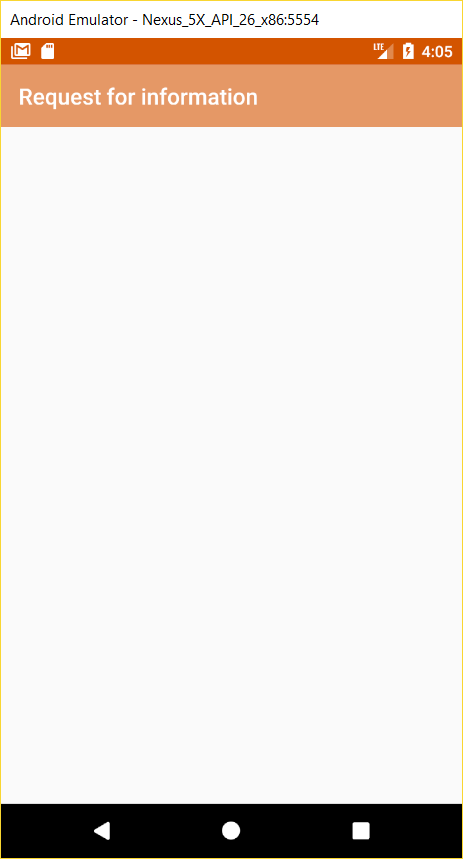
default:

return super.onOptionsItemSelected(item);

}

}

Run the app:

## Enabling Up navigation

You can enable the *Up* button on the app bar to let users navigate through the app using hierarchical relationships. We can enable the *Up* button on *InfoRequestActivity*’s app bar so that the user can return to *MainActivity* by clicking on this button.

The first step is to set the activity’s parent in *AndroidManifest.xml* (see highlighted text below):

12. parentActivityName.txt

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="ca.bcit.appbarmania">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:roundIcon="@mipmap/ic\_launcher\_round"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<activity

android:name=".InfoRequestActivity"

android:parentActivityName=".MainActivity"

android:label="@string/info\_request" >

</activity>

</application>

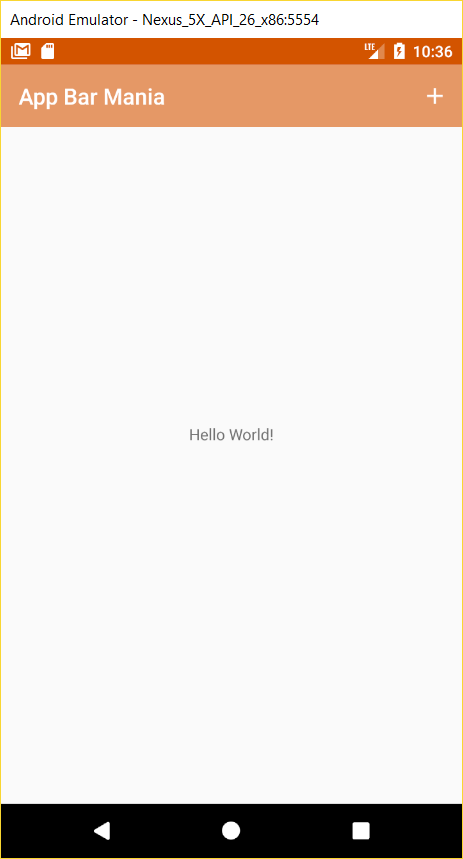
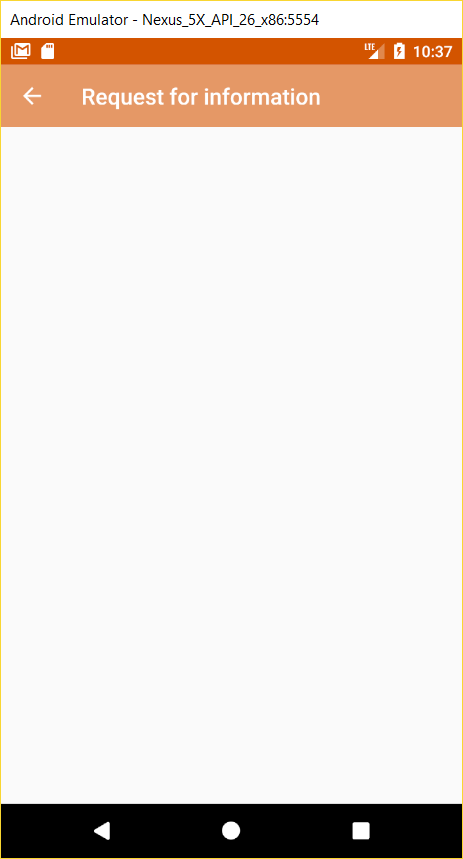
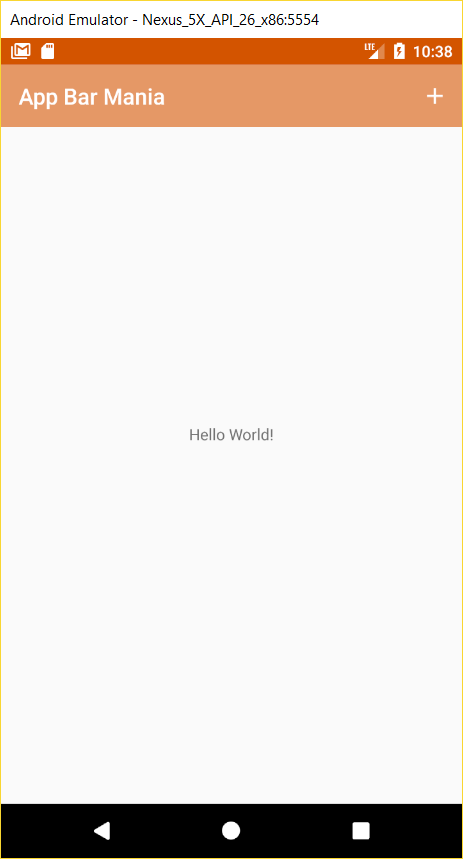
</manifest>

Next, we need will enable the Up button from within the code for *InfoRequestActivity* by adding the following code to the bottom of *onCreate()*:

13. onCreate\_ActionBar.txt

ActionBar ab = getSupportActionBar();

ab.setDisplayHomeAsUpEnabled(true);

## Sharing content on the app bar

Let us add a share action provider to the app. This allows users to share content on the app with other apps such as Gmail. The share action provider defines its own icon.

Add the following to strings.xml:

14. action\_share.txt

<string name="action\_share">Share</string>

Add the following menu item to *menu\_main.xml*:

15. menu\_item\_action\_share.txt

<item android:id="@+id/action\_share"

android:title="@string/action\_share"

android:orderInCategory="2"

app:showAsAction="ifRoom"

app:actionProviderClass="android.support.v7.widget.ShareActionProvider"/>

Add the following instance variable to *MainActivity*:

16. private\_ShareActionProvider.txt

private ShareActionProvider shareActionProvider;

Get a reference to the share action provider and assign it to the private variable. Add this method to *MainActivity*:

17. setShareActionIntent.txt

private void setShareActionIntent(String text) {

Intent i = new Intent(Intent.ACTION\_SEND);

i.setType("text/plain");

i.putExtra(Intent.EXTRA\_TEXT, text);

shareActionProvider.setShareIntent(i);

}

Add this code to *onCreateOptionsMenu()* in *MainActivity,* just before the return statement:

18. MainActivity\_onCreateOptionsMenu.txt

MenuItem menuItem = menu.findItem(R.id.action\_share);

shareActionProvider = (ShareActionProvider) MenuItemCompat.getActionProvider(menuItem);

setShareActionIntent("Join us on this vacation.");

Run the app:

