12. Menus

When we created the project, Android SDK created required methods for us in order to program options menu. These methods were located in MainActivity. Our task here is to take exit button to options menu. Therefore let’s look how options menu can be programmed.
onCreateOptionsMenu(Menu menu) inflates the menu xml file which is located in Res > menu > main.xml

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

The second method is where we define what happens when an item from option menu is clicked.

```
@Override
      public boolean onOptionsItemSelected(MenuItem item) {
            // Handle action bar item clicks here. The action bar
will
            // automatically handle clicks on the Home/Up button, so
long
            // as you specify a parent activity in
AndroidManifest.xml.
            int id = item.getItemId();
            if (id == R.id.action_settings) {
                  return true;
            }
            return super.onOptionsItemSelected(item);
      }
```

Finally we will edit the main.xml file to add, edit or remove menu items. Here you can add more items and use their id as a reference to find out when that particular item is clicked.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="fi.metropolia.qaisersiddique.MainActivity" >

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never"/>

</menu>
```

13. Supporting Multiple Screen Sizes (Android.com, 2014)

Android categorizes device screens using two general properties: size and density. You should expect that your app will be installed on devices with screens that range in both size and density. As such, you should include some alternative resources that optimize your app’s appearance for different screen sizes and densities.

- There are four generalized sizes: small, normal, large, xlarge
- And four generalized densities: low (ldpi), medium (mdpi), high (hdpi), extra high (xhdpi)

To declare different layouts and bitmaps you'd like to use for different screens, you must place these alternative resources in separate directories, similar to how you do for different language strings.
Also be aware that the screens orientation (landscape or portrait) is considered a variation of screen size, so many apps should revise the layout to optimize the user experience in each orientation.

13.1 Create Different Layouts

To optimize your user experience on different screen sizes, you should create a unique layout XML file for each screen size you want to support. Each layout should be saved into the appropriate resources directory, named with a -<screen_size> suffix. For example, a unique layout for large screens should be saved underres/layout-large/.
Note: Android automatically scales your layout in order to properly fit the screen. Thus, your layouts for different screen sizes don't need to worry about the absolute size of UI elements but instead focus on the layout structure that affects the user experience (such as the size or position of important views relative to sibling views).
For example, this project includes a default layout and an alternative layout for *large* screens:

```
MyProject/
    res/
        layout/
            main.xml
        layout-large/
            main.xml
```

The file names must be exactly the same, but their contents are different in order to provide an optimized UI for the corresponding screen size.
Simply reference the layout file in your app as usual:

```
@Override
 protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
}
```

The system loads the layout file from the appropriate layout directory based on screen size of the device on which your app is running. More information about how Android selects the appropriate resource is available in the Providing Resources guide.
As another example, here's a project with an alternative layout for landscape orientation:

```
MyProject/
    res/
        layout/
            main.xml
        layout-land/
            main.xml
```

By default, the layout/main.xml file is used for portrait orientation.
If you want to provide a special layout for landscape, including while on large screens, then you need to use both the large and land qualifier:

```
MyProject/
    res/
        layout/              # default (portrait)
            main.xml
        layout-land/         # landscape
            main.xml
        layout-large/        # large (portrait)
            main.xml
        layout-large-land/   # large landscape
            main.xml
```

Note: Android 3.2 and above supports an advanced method of defining screen sizes that allows you to specify resources for screen sizes based on the minimum width and height in terms of density-independent pixels. This lesson does not cover this new technique. For more information, read Designing for Multiple Screens.

13.2 Create Different Bitmaps

You should always provide bitmap resources that are properly scaled to each of the generalized density buckets: low, medium, high and extra-high density. This helps you achieve good graphical quality and performance on all screen densities.
To generate these images, you should start with your raw resource in vector format and generate the images for each density using the following size scale:

- xhdpi: 2.0
- hdpi: 1.5
- mdpi: 1.0 (baseline)
- ldpi: 0.75

This means that if you generate a 200x200 image for xhdpi devices, you should generate the same resource in 150x150 for hdpi, 100x100 for mdpi, and 75x75 for ldpi devices.
Then, place the files in the appropriate drawable resource directory:

```
MyProject/
    res/
        drawable-xhdpi/
            awesomeimage.png
        drawable-hdpi/
            awesomeimage.png
        drawable-mdpi/
            awesomeimage.png
        drawable-ldpi/
            awesomeimage.png
```

Any time you reference **@drawable/awesomeimage**, the system selects the appropriate bitmap based on the screen's density.

Note: Low-density (ldpi) resources aren't always necessary. When you provide hdpi assets, the system scales them down by one half to properly fit ldpi screens.
For more tips and guidelines about creating icon assets for your app, see the Iconography design guide.