UNIT 7 DEVICE COMPATIBILITY

7.0 INTRODUCTION

Android is designed to run on many different types of devices, from phones to tablets and televisions. The range of devices provides a huge potential audience for the Android applications. In order to be successful on all these devices, it provides a flexible user interface that adapts to different screen configurations.

Android provides a dynamic app framework that can provide configuration-specific app resources in static files such as different XML layouts for different screen sizes. Android loads the appropriate resources based on the current device configuration. With some additional app resources, developer can publish a single application package (APK) that provides an optimized user experience on a variety of devices.

This unit will be focused on device compatibility. There are two types of compatibility *device compatibility* and *app compatibility*.

Hardware manufacturer can build a device that runs the Android operating system. Yet, a device is **"Android compatible"** only if it can correctly run apps written for the *Android execution environment* and each device must pass the Compatibility Test Suite (CTS) in order to be considered compatible.

Though Android runs on a wide range of device configurations, some features are not available on all devices. For example, some devices may not include a compass sensor. If your app's core functionality requires the use of a compass sensor, then your app is compatible only with devices that include a compass sensor.

7.1 OBJECTIVES

After studying this unit, you should be able to:

Outcomes

- identify compatibility of an application with different devices.
- discuss screen configuration for various device sizes and resolutions
- evaluate device compatibility of an application

Terminology

Widget:	an application, or a component of an interface, that enables a user to perform a function or access a service
platform:	where any piece of software is executed
screen density:	quantity of pixels within a physical area of the screen
resolution:	The total number of physical pixels on a screen

7.2 APPLICATION AVAILABILITY TO DEVICES

Android supports a variety of features your app can control through platform APIs. Some features are hardware-based such as a compass sensor, some are software-based such as app widgets, and some features dependent on the platform version. You have to control application availability to the devices based on the features of your application because not every device supports every feature.

To achieve the largest user-base possible for an app, developer should strive to support as many device configurations as possible using a single APK. In most situations, it can do by disabling optional features at runtime and providing app resources with alternatives for different configurations.

Device characteristics are Device features, Platform version and Screen configuration.

7.2.1 Check Your Progress

State the importance of device configuration to maintain device compatibility

7.3 DEVICES FEATURES

In order to manage your app's availability based on device features, Android defines *feature IDs* for any hardware or software feature that may not be available on all devices.

For instance, you can prevent users from installing your app when their devices do not provide a given feature by declaring it with a <uses-feature> element in your app's manifest file.

Example: Declare the compass sensor

If your app does not make sense on a device that lacks a compass sensor, you can declare the compass sensor as required with the following manifest tag as shown in the code snippet below.

```
<manifest ... >
    <uses-featureandroid:name="android.hardware.sensor.compass"
                  android:required="true"/>
    ...
</manifest>
```

Google Play Store compares the features that your app requires to the features available on each user's device to determine whether your app is compatible with that device. If the device does not provide all the features your app requires, the user cannot install your app.

However, if your app's primary functionality does not require a device feature, you should set the required attribute to "false" and check for the device feature at runtime. If the app feature is not available on the current device, gracefully degrade the corresponding app feature. For example, you can query whether a feature is available by calling hasSystemFeature() like this:

```
PackageManager pm = getPackageManager();
if(!pm.hasSystemFeature(PackageManager.FEATURE_SENSOR_COMPASS))
{
// This device does not have a compass, turn off the  //compass
feature
    disableCompassFeature();
}
```

7.4 PLATFORM VERSION

Different devices may run different versions of the Android platform, such as Android 4.0 or Android 6.0. Each successive platform version often adds new APIs not available in the previous version. To indicate which set of APIs are available, each platform version specifies an API level.

For instance, Android 1.0 is API level 1 and Android 6.0 is API level 23

The API level allows you to declare the minimum version with which your app is compatible, using the <uses-sdk>manifest tag and its minSdkVersion attribute.

For example:
The Calendar Provider APIs were added in Android 4.0 (API level 14). If your app cannot function without these APIs, you should declare API level 14 as your app's minimum supported version like this:

```
<manifest ... >
    <uses-
sdkandroid:minSdkVersion="14"android:targetSdkVersion="19"/>
    ...
</manifest>
```

The minSdkVersion attribute declares the minimum version with which your app is compatible and the targetSdkVersion attribute declares the highest version on which you have optimized your app. Each successive version of Android provides compatibility for apps that were built using the APIs from previous platform versions, so your app should always be compatible with future versions of Android while using the documented Android APIs.

However, if your app uses APIs added in a more recent platform version, but does not require them for its primary functionality, you should check the API level at runtime and gracefully degrade the corresponding features when the API level is too low.

In this case, set the minSdkVersion to the lowest value possible for your app's primary functionality, then compare the current system's version, SDK_INT, to one the codename constants in

Build.VERSION_CODES that corresponds to the API level you want to check.

For example:

```
if(Build.VERSION.SDK_INT <Build.VERSION_CODES.HONEYCOMB){
    // Running on something older than API level 11, so disable
    // the drag/drop features that use ClipboardManager APIs
    disableDragAndDrop();
}
```

7.4.1 Check Your Progress

Discuss different devices and different versions of the Android platform

Now we will see how Android runs on devices of various sizes.

7.5 SCREEN CONFIGURATION

Android runs on devices of various sizes, from phones to tablets and TVs. In order to categorize devices by their screen type, Android defines characteristics for each device:

- **Screen size -**The physical size of the screen. Actual physical size, measured as the screen's diagonal.
- For simplicity, Android groups all actual screen sizes into four generalized sizes: small, normal, large, and extra-large.
- **Screen density** -. The quantity of pixels within a physical area of the screen; usually referred to as dpi (dots per inch). For example, a "low" density screen has fewer pixels within a given physical area, compared to a "normal" or "high" density screen.
- For simplicity, Android groups all actual screen densities into six generalized densities: low, medium, high, extra-high, extra-extra-high, and extra-extra-extra-high.
- **Resolution**- The total number of physical pixels on a screen. When adding support for multiple screens, applications do not work directly with resolution; applications should be concerned only with screen size and density, as specified by the generalized size and density groups.

A set of six generalized densities

- ldpi (low) ~120dpi
- mdpi (medium) ~160dpi
- hdpi (high) ~240dpi
- xhdpi (extra-high) ~320dpi
- xxhdpi (extra-extra-high) ~480dpi

Density-independent pixel (dp)

A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way.

The density-independent pixel is equivalent to one physical pixel on a 160 dpi screen, which is the baseline density assumed by the system for a "medium" density screen.

At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use. The conversion of dp units to screen pixels is simple: px = dp * (dpi / 160).

For example, on a 240 dpi screen, 1 dp equals 1.5 physical pixels. You should always use dp units when defining your application's UI, to ensure proper display of your UI on screens with different densities.

By default, your app is compatible with all screen sizes and densities, because the system makes the appropriate adjustments to your UI layout and image resources as necessary for each screen. However, you should optimize the user experience for each screen configuration by adding specialized layouts for different screen sizes and optimized bitmap images for common screen densities.

Use wrap_content and match_parent

To ensure that your layout is flexible and adapts to different screen sizes, you should use "wrap_content" and "match_parent" for the width and height of some view components. If you use "wrap_content", the width or height of the view is set to the minimum size necessary to fit the content within that view, while "match_parent" makes the component expand to match the size of its parent view. By using the "wrap_content" and "match_parent" size values instead of hard-coded sizes, your views either use only the space required for that view or expand to fill the available space, respectively.

For example:

```
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/
android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayoutandroid:layout_width="match_parent"
                  android:id="@+id/linearLayout1"
                  android:gravity="center"
                  android:layout_height="50dp">
        <ImageViewandroid:id="@+id/imageView1"
                   android:layout_height="wrap_content"
                   android:layout_width="wrap_content"
                   android:src="@drawable/logo"
```

```
                    android:paddingRight="30dp"
                    android:layout_gravity="left"
                    android:layout_weight="0"/>
        <Viewandroid:layout_height="wrap_content"
                android:id="@+id/view1"
                android:layout_width="wrap_content"
                android:layout_weight="1"/>
        <Buttonandroid:id="@+id/categorybutton"
                  android:background="@drawable/button_bg"
                  android:layout_height="match_parent"
                  android:layout_weight="0"
                  android:layout_width="120dp"
                  style="@style/CategoryButtonStyle"/>
    </LinearLayout>

    <fragmentandroid:id="@+id/headlines"
                android:layout_height="fill_parent"
                android:name="com.example.android.newsreader.Head
linesFragment"
                android:layout_width="match_parent"/>
</LinearLayout>
```

Notice how the sample uses "wrap_content" and "match_parent" for component sizes rather than specific dimensions. This allows the layout to adapt correctly to different screen sizes and orientations.

For example, figure 7.1 shows what this layout looks like in portrait and landscape mode. Notice that the sizes of the components adapt automatically to the width and height:

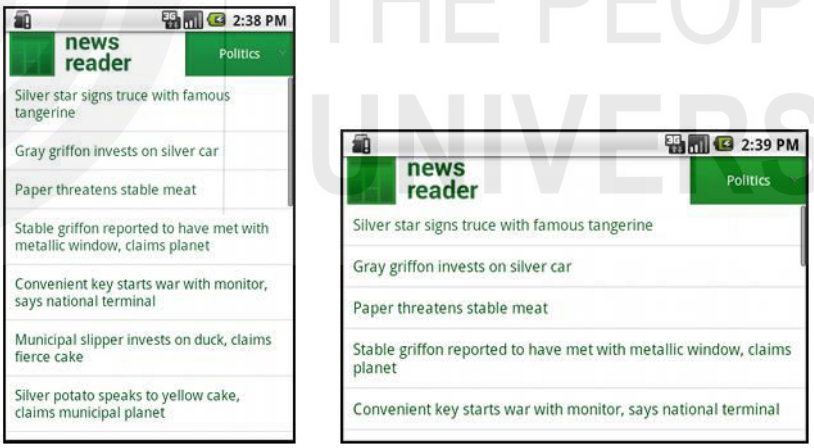

Figure 7.1: News Reader sample app in portrait (left) and landscape (right) (Source:https://developer.android.com/training/multiscreen/screensizes.html)

7.5.1 Video- V7: Device Compatibility

Let us watch the video on device compatibility and do the activity 7.3.

URL: https://tinyurl.com/ydcl9trg

7.5.2 Check Your Progress

Calculate resolution values for your mobile device using density independent pixels.

7.6 SUMMARY

In this unit, we discussed device compatibility and application availability to devices based on the device characteristics. These characteristics include device features, Platform version and Screen configuration. Furthermore, Android defines characteristics for each device such as screen size, density and resolution.

7.7 FURTHER READINGS

https://developer.android.com/guide/app-compatibility

https://developer.android.com/guide/practices/screens_support