MCA253 - Mobile Applications

Unit: 2.0
Getting to Know the Android User Interface

Dr. Siddesha S MCA, MISTE, M.Sc Tech (by Research) , Ph.D.
Asst. Professor,
Dept. of Computer Applications,
JSS Science and Technology University
Mysuru – 570 006

Understanding the Components of a Screen

- The basic unit of an Android application is an activity, which displays the UI of your application.
- The activity may contain widgets such as buttons, labels, textboxes, and so on.
- Typically, UI is defined using an XML file (for example, the activity_main.xml file located in the res/layout folder of the current project).
- During runtime, XML UI loaded in the onCreate() method handler in Activity class, using the setContentView() method of the Activity class.
- During compilation, each element in the XML file is compiled into its equivalent Android GUI (Graphical User Interface) class, with attributes represented by methods.
- The Android system then creates the activity's UI when the activity is loaded.

Views and ViewGroups

- An activity contains *views* and *ViewGroups*.
- A view is a widget that has an appearance on screen.
- Examples of views are buttons, labels, and text boxes.
- A view derives from the base class android.view.View.
- One or more views can be grouped into a ViewGroup.
- A ViewGroup (which is itself a special type of view) provides the layout in which you can order the appearance and sequence of views.
- Examples of ViewGroups include RadioGroup and ScrollView.
- A ViewGroup derives from the base class android.view.ViewGroup.

Views and ViewGroups

- Another type of ViewGroup is a Layout.
- A Layout is another container that derives from android.view.ViewGroup and is used as a container for other views.
- However, whereas the purpose of a ViewGroup is to group views logically—such as a group of buttons with a similar purpose—a Layout is used to group and arrange views visually on the screen.
- Different layouts available in Android are
  - FrameLayout
  - LinearLayout(Horizontal)
  - LinearLayout(Vertical)
  - TableLayout
  - TableRow
  - GridLayout
  - RelativeLayout

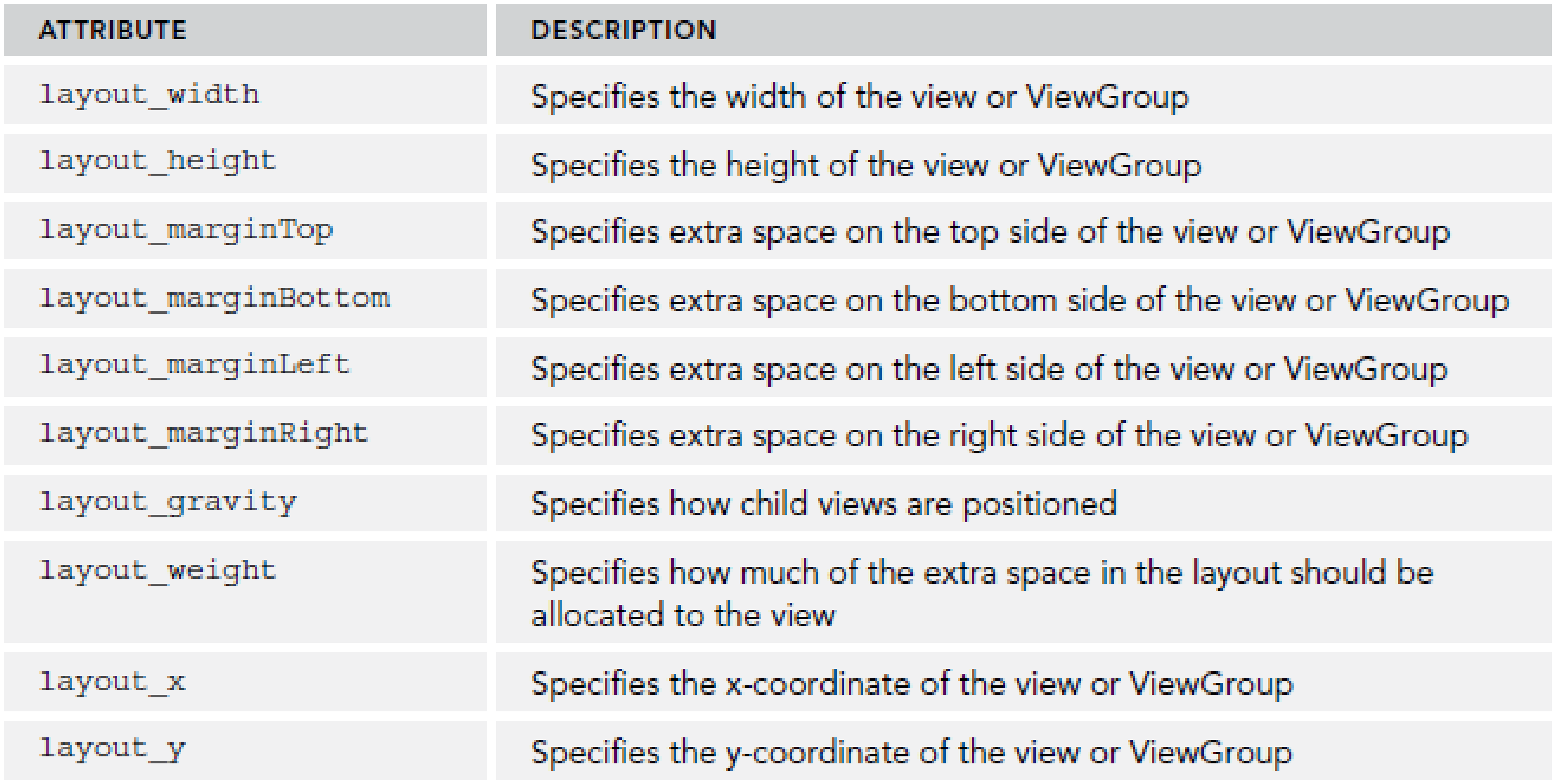
FrameLayout

- The FrameLayout is the most basic of the Android layouts.
- FrameLayouts are built to hold one view.
- As with all things related to development, there is no hard rule that FrameLayouts can’t be used to hold multiple views.
- However, there is a reason why FrameLayouts were built the way they were.
- Given that there are myriad screen sizes and resolutions, you have little control over the specifications of the devices that install your application.
- Therefore, when your application is resized and reformatted to fit any number of different devices you want to make sure it still looks as close to your initial design as possible.
- The FrameLayout is used to help you control the stacking of single views as the screen is resized.

LinearLayout(Horizontal) & LinearLayout(Vertical)

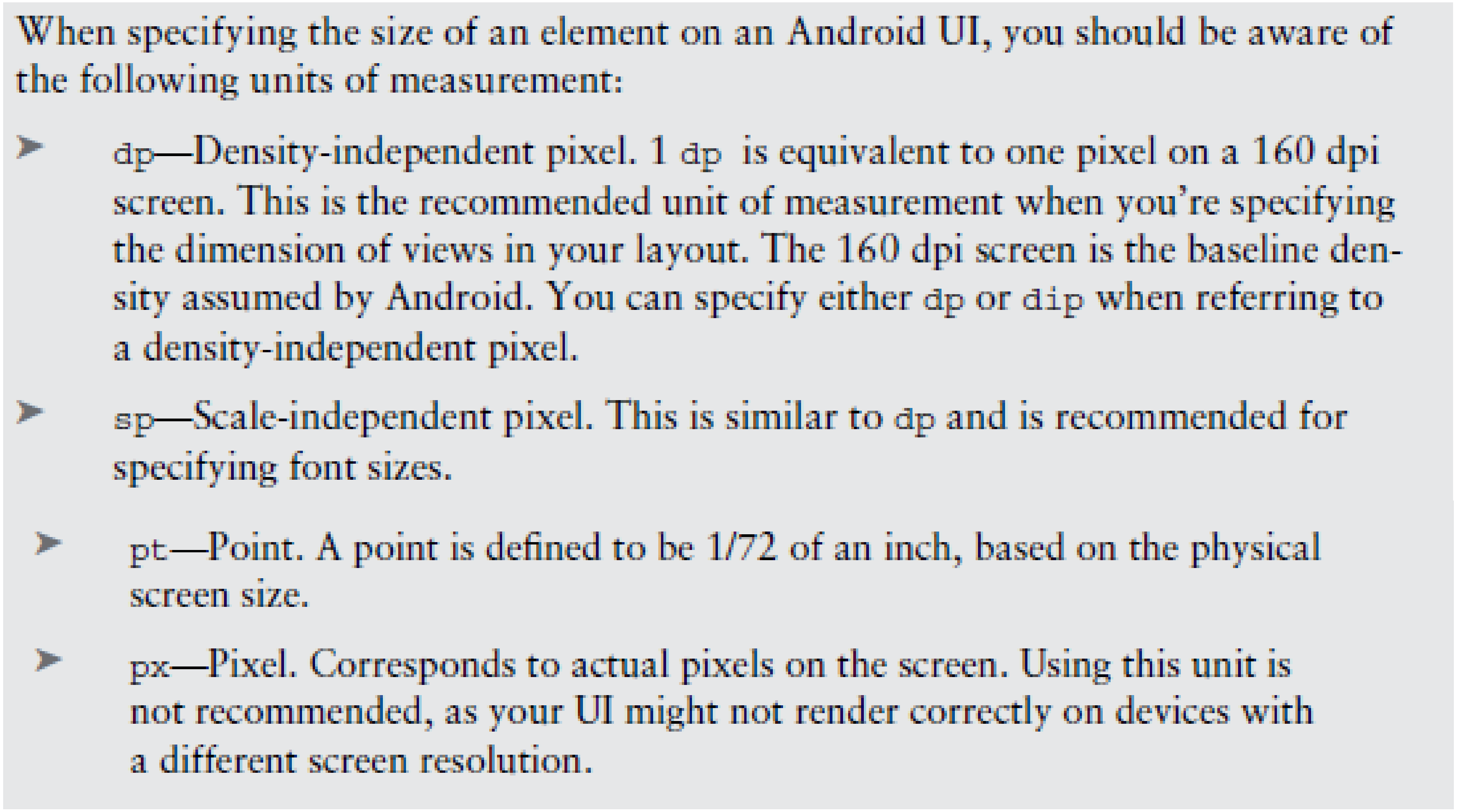
- The LinearLayout arranges views in a single column or a single row.
- Child views can be arranged either horizontally or vertically, which explains the need for two different layouts—one for horizontal rows of views and one for vertical columns of views.
- In LinearLayout, you can apply the layout_weight and layout_gravity attributes to views contained within it, as the modifications to activity_main.xml
- If you change the orientation of the LinearLayout to horizontal you need to change the width of each view to 0 dp

LinearLayout(Horizontal) & LinearLayout(Vertical)



ATTRIBUTE	DESCRIPTION
`layout_width`	Specifies the width of the view or ViewGroup
`layout_height`	Specifies the height of the view or ViewGroup
`layout_marginTop`	Specifies extra space on the top side of the view or ViewGroup
`layout_marginBottom`	Specifies extra space on the bottom side of the view or ViewGroup
`layout_marginLeft`	Specifies extra space on the left side of the view or ViewGroup
`layout_marginRight`	Specifies extra space on the right side of the view or ViewGroup
`layout_gravity`	Specifies how child views are positioned
`layout_weight`	Specifies how much of the extra space in the layout should be allocated to the view
`layout_x`	Specifies the x-coordinate of the view or ViewGroup
`layout_y`	Specifies the y-coordinate of the view or ViewGroup

Units of Measurements



When specifying the size of an element on an Android UI, you should be aware of the following units of measurement:

- dp—Density-independent pixel. 1 dp is equivalent to one pixel on a 160 dpi screen. This is the recommended unit of measurement when you're specifying the dimension of views in your layout. The 160 dpi screen is the baseline density assumed by Android. You can specify either dp or dip when referring to a density-independent pixel.
- sp—Scale-independent pixel. This is similar to dp and is recommended for specifying font sizes.
- pt—Point. A point is defined to be 1/72 of an inch, based on the physical screen size.
- px—Pixel. Corresponds to actual pixels on the screen. Using this unit is not recommended, as your UI might not render correctly on devices with a different screen resolution.

Two LinearLayouts with Different Orientations

- Use two LinearLayouts, one with a vertical orientation and one with a horizontal orientation to place three TextViews and three Buttons.

Program: LinearLayout

TableLayout

- The TableLayout Layout groups views into rows and columns.
- <TableRow> element is used to designate a row in the table.
- Each row can contain one or more views.
- Each view you place within a row forms a cell.
- The width of each column is determined by the largest width of each cell in that column.

Program: TableLayout

RelativeLayout

- The RelativeLayout layout enables you to specify how child views are positioned relative to each other.

Program: RelativeLayout

FrameLayout

- The FrameLayout layout is a placeholder on screen that you can use to display a single view.
- Views that add to a FrameLayout are always anchored to the top left of the layout.

Program: FrameLayoutNew & FrameLayout2 - with image view

ScrollView

- A ScrollView is a special type of FrameLayout in that it enables users to scroll through a list of views that occupy more space than the physical display.
- The ScrollView can contain only one child view or ViewGroup, which normally is a LinearLayout.

Program: ScrollView

Adapting to Display Orientation

- One of the key features of modern smartphones is their ability to switch screen orientation, and Android is no exception.
- Android supports two screen orientations: portrait and landscape.
- By default, when you change the display orientation of your Android device, the current activity automatically **redraws** its content in the new orientation.
- This is because the onCreate() method of the activity is fired whenever there is a change in display orientation.
- However, when the views are redrawn, they may be drawn in their original locations (depending on the layout selected)

Adapting to Display Orientation

- In general, you can employ two techniques to handle changes in screen orientation:
  - **Anchoring**—The easiest way is to “anchor” your views to the four edges of the screen.
  - When the screen orientation changes, the views can anchor neatly to the edges.
  - **Resizing and repositioning**—Whereas anchoring and centralizing are simple techniques to ensure that views can handle changes in screen orientation, the ultimate technique is resizing each and every view according to the current screen orientation.

Program: AnchorView

Managing Changes to Screen Orientation

Program: ScreenOrientation

Persisting State Information During Changes in Configuration

- Keep in mind that when an activity is re-created, its current state might be lost. When an activity is killed, it fires one or both of the following methods:
  - onPause()—This method is always fired whenever an activity is killed or pushed into the background.
  - onSaveInstanceState()—This method is also fired whenever an activity is about to be killed or put into the background (just like the onPause() method). However, unlike the onPause() method, the onSaveInstanceState() method is not fired when an activity is being unloaded from the stack (such as when the user pressed the back button) because there is no need to restore its state later.

Persisting State Information During Changes in Configuration

- In short, to preserve the state of an activity, you could always implement the onPause() method and then use your own ways to preserve the state of your activity, such as using a database, internal or external file storage, and so on.
- If you simply want to preserve the state of an activity so that it can be restored later when the activity is re-created (such as when the device changes orientation), a much simpler way is to implement the onSaveInstanceState() method, as it provides a Bundle object as an argument so that you can use it to save your activity’s state.
- The following code shows that you can save the string ID into the Bundle object during the onSaveInstanceState() method:

Persisting State Information During Changes in Configuration

```
@Override
public void onSaveInstanceState(Bundle outState) {
//---save whatever you need to persist---
outState.putString("ID", "1234567890");
super.onSaveInstanceState(outState);
}
```

- When an activity is re-created, the onCreate() method is first fired, followed by the onRestoreInstanceState() method, which enables you to retrieve the state that you saved previously in the onSaveInstanceState() method through the Bundle object in its argument.

```
@Override
public void onRestoreInstanceState(Bundle savedInstanceState)
{
     super.onRestoreInstanceState(savedInstanceState);
    //---retrieve the information persisted earlier---
     String ID = savedInstanceState.getString("ID");
}
}
```

Detecting Orientation Changes

- Sometimes it is needed to know the device’s current orientation during runtime.
- To determine that, getResources() method can be used.
- The following code snippet demonstrates how you can programmatically detect the current orientation of your activity,

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   if(getResources().getConfiguration().orientation ==
      Configuration.ORIENTATION_LANDSCAPE){
         Log.d("StateInfo", "Landscape");}
   else if(getResources().getConfiguration().orientation ==
      Configuration.ORIENTATION_PORTRAIT){
Log.d("StateInfo", "Portrait"); }
}
```

Controlling the Orientation of the Activity

- Occasionally, it is needed to ensure that the application is displayed in only a certain orientation.
- For example, while writing a game that should be viewed only in landscape mode.
- In this case, user can programmatically force a change in orientation using the setRequestOrientation() method of the Activity class:

```
import android.content.pm.ActivityInfo;
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    //---change to landscape mode---
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
}
```

- To change to portrait mode, use the `ActivityInfo.SCREEN_ORIENTATION_PORTRAIT` constant

Controlling the Orientation of the Activity

- Besides using the setRequestOrientation() method, you can also use the android:screenOrientation attribute on the <activity> element in AndroidManifest.xml as follows to constrain the activity to a certain orientation:

```
<activity android:name=".MainActivity"
    android:screenOrientation="landscape" >
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
 </activity>
```

- Two other values that you can specify in the android:screenOrientation attribute:
  - ❑ portrait—Portrait mode
  - ❑ sensor—Based on the accelerometer (default)

Utilizing the Action Bar

- Besides fragments, another feature of Android is the Action Bar. In place of the traditional title bar located at the top of the device's screen, the Action Bar displays the application icon and the activity title.
- Optionally, on the right side of the Action Bar are action items.

Showing and Hiding the Action Bar:

- Create a new android application and run the app in emulator, name of the application is displayed on the action bar eg. If your app name is HelloWorld, then you can see HelloWorld on the action bar.
- When you are creating an App some times this looks odd, so to hide the content on the Action Bar programmatically, remove the following statement in the MainActivity.java. Within the Oncreate() method.

```
setSupportActionBar(toolbar);
```

Adding Action Items to the Action Bar

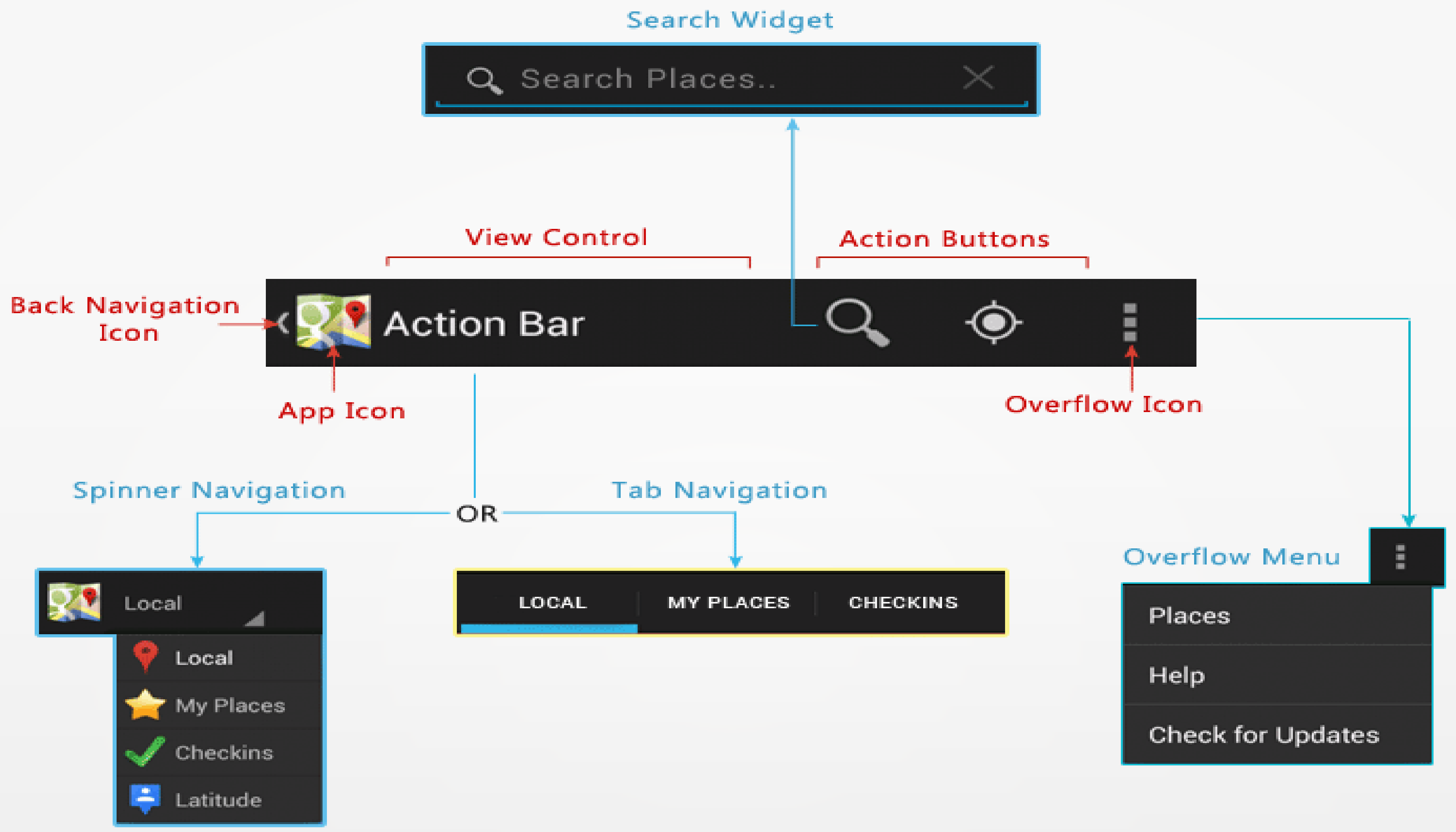
- Besides displaying the application icon and the activity title on the left of the Action Bar, you can also display additional items on the Action Bar.
- These additional items are called action items.
- *Action items* are shortcuts to some of the commonly performed operations in your application.
- For example, you might be building an RSS reader application, in which case some of the action items might be Refresh Feed, Delete Feed, and Add New Feed.

Overview of Action Bar

- Action bar mainly contains four functional areas.
- They are app icon, view control, action buttons and action overflow.
- App Icon – App branding logo or icon will be displayed here.
- View Control – A dedicated space to display app title. Also provides option to switch between views by adding spinner or tabbed navigation.
- Action Buttons – Some important actions of the app can be added here.
- Action Overflow – All unimportant action will be shown as a menu.

Programs: FruitMenu and OptionMenu

Android Action Bar Overview



Creating the User Interface Programmatically

- So far, all the UIs you have seen are created using XML.
- As mentioned earlier, besides using XML you can also create the UI using code.
- This approach is useful if your UI needs to be dynamically generated during runtime.
- For example, suppose you are building a cinema ticket reservation system and your application displays the seats of each cinema using buttons.
- In this case, you need to dynamically generate the UI based on the cinema selected by the user.

Program: CreateUIProgram

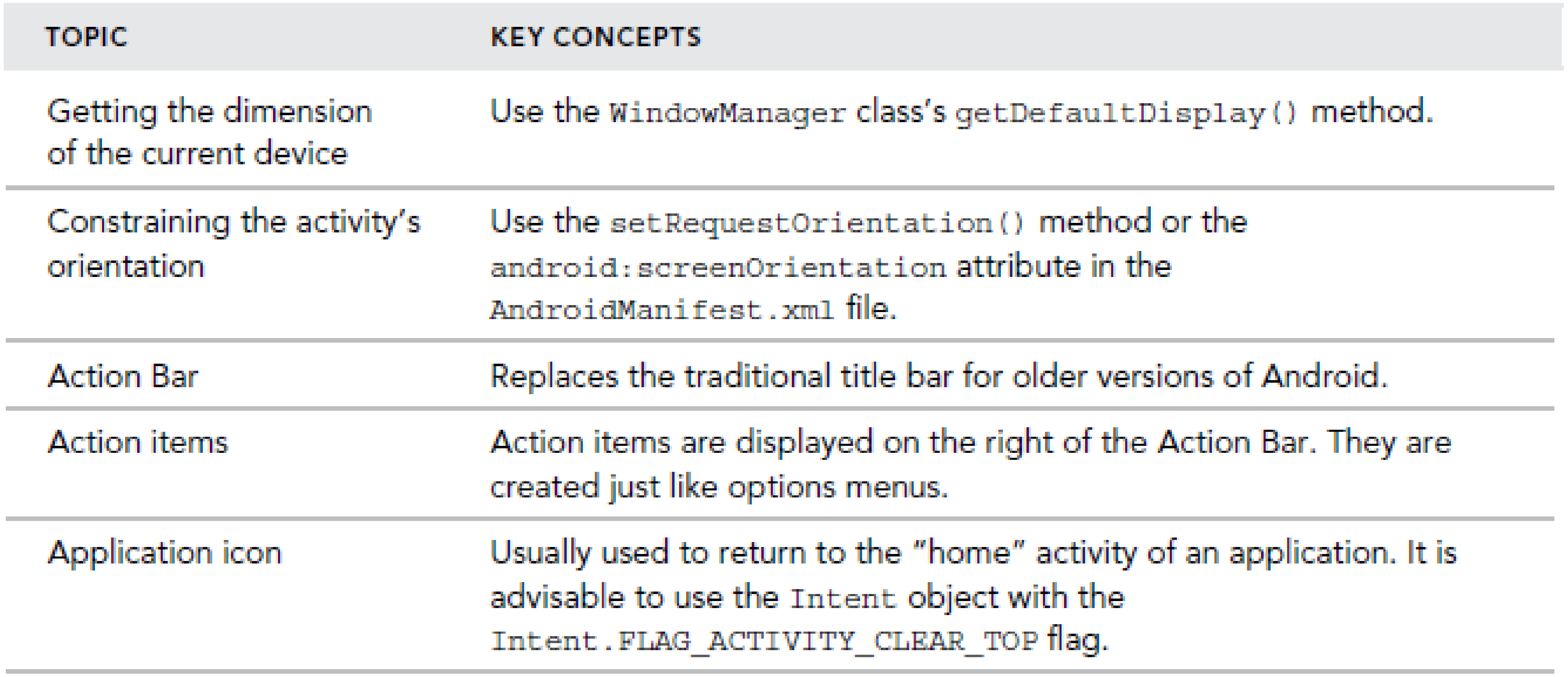
Listening for UI Notifications

- Users interact with your UI at two levels: the activity level and the view level. At the activity level, the Activity class exposes methods that you can override. Some common methods that you can override in your activities include the following:
  - onKeyDown —Called when a key was pressed and not handled by any of the views contained within the activity
  - onKeyUp —Called when a key was released and not handled by any of the views contained within the activity
  - onMenuItemSelected —Called when a panel's menu item has been selected by the user
  - onMenuOpened —Called when a panel's menu is opened by the user

Summary

TOPIC	KEY CONCEPTS
`LinearLayout`	Arranges views in a single column or single row.
`AbsoluteLayout`	Enables you to specify the exact location of its children.
`TableLayout`	Groups views into rows and columns.
`RelativeLayout`	Enables you to specify how child views are positioned relative to each other.
`FrameLayout`	A placeholder on screen that you can use to display a single view.
`ScrollView`	A special type of `FrameLayout` in that it enables users to scroll through a list of views that occupy more space than the physical display allows.
Unit of Measure	Use `dp` for specifying the dimension of views and `sp` for font size.
Two ways to adapt to changes in orientation	Anchoring, and resizing and repositioning.
Using different XML files for different orientations	Use the `layout` folder for portrait UI, and `layout-land` for landscape UI.
Three ways to persist activity state	Use the `onPause()` method. Use the `onSaveInstanceState()` method. Use the `onRetainNonConfigurationInstance()` method.

Summary



TOPIC	KEY CONCEPTS
Getting the dimension of the current device	Use the `WindowManager` class's `getDefaultDisplay()` method.
Constraining the activity's orientation	Use the `setRequestOrientation()` method or the `android:screenOrientation` attribute in the `AndroidManifest.xml` file.
Action Bar	Replaces the traditional title bar for older versions of Android.
Action items	Action items are displayed on the right of the Action Bar. They are created just like options menus.
Application icon	Usually used to return to the "home" activity of an application. It is advisable to use the `Intent` object with the `Intent.FLAG_ACTIVITY_CLEAR_TOP` flag.

Program: MyNotification