Australian National University

COMP2100/6442
Software Design Methodologies / Software Construction

Android

Bernardo Pereira Nunes

slides based on https://developer.android.com/guide

Android Fundamentals

Android apps
 - Kotlin, Java, and C++ languages.

After compiled, all data and resources are stored in an APK (Android Package). APK is the file used to install the app on your phone! (e.g., blah.apk)

Android OS is a multi-user Linux system (each app is a different user)

Security – App's code runs in isolation from other apps; system assigns an ID for all files (only the user id can access the files)

Principle of Least Privilege – each app, by default, has access only to the components that it requires to do its work and no more. (secure environment, app cannot access other parts of the system without permission)

However, apps can ask permission to access other resources, for instance, camera, gps location, etc.

Android App

> Android apps are built as a combination of components that can be invoked individually. For example, an **activity is a type of app component that provides a user interface (UI).**

> The "main" activity starts when the user taps your app's icon. You can also direct the user to an activity from elsewhere, such as from a notification or even from a different app.

App components – Building blocks

Each component is an entry point through which the system or a user can enter your app. Some components depend on others.

There are four different types of app components:

Activities - entry point for interacting with the user! Single screen with a user interface.

Services - a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. (e.g., play music in the background while user is in another app)

Broadcast receivers - is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements. (e.g., screen has turned off, the battery is low, or a picture was captured)

Content providers - manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access.

Australian National University

Intent

An Intent object carries information that the Android system uses **to determine which component to start** plus **information that the recipient component uses** in order to properly **perform the action** (such as the action to take and the data to act upon).

```
Intent intent = new Intent(getApplicationContext(), ActivityB.class);
startActivity(intent);
```

context for the entire application

Redirect current activity to another (ActivityB.class)

Perform intent!

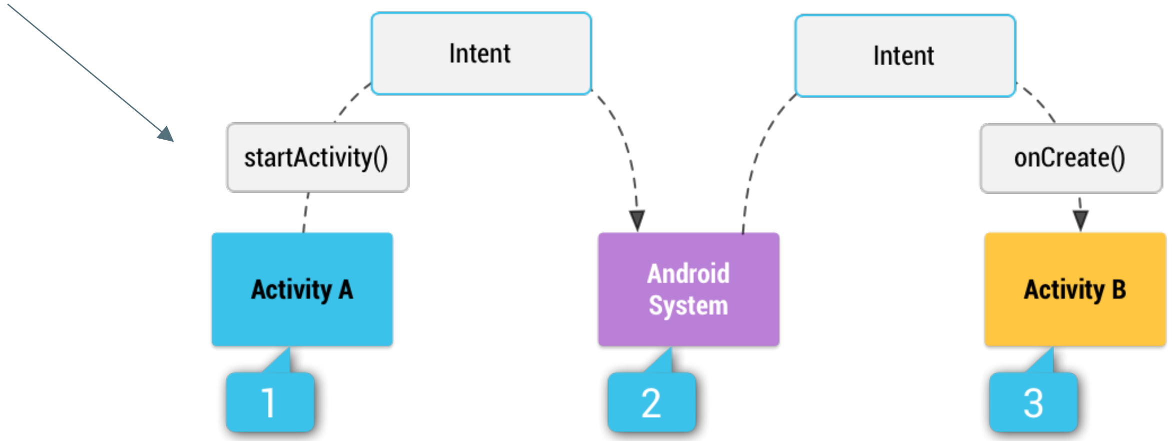
Australian National University

Activating components (Intent)

Activities, services, and broadcast receivers - are activated by an asynchronous message called an intent. Intents bind individual components to each other at runtime. There are two types of intents: implicit and explicit.

Explicit: use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start.

Implicit: do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, use an implicit intent to request another capable app to do something (show user location in another app)



Activities

The Activity class is a crucial component of an Android app.

Unlike programming paradigms in which apps are launched with a main() method, the Android system initiates code in an Activity instance by invoking specific callback methods that correspond to specific stages of its lifecycle.

An activity provides the window in which the app draws its UI.

Typically, one activity in an app is specified as the main activity, which is the first screen to appear when the user launches the app.

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View).

Australian National University

Activities

There are **two methods almost all subclasses of Activity will implement**:

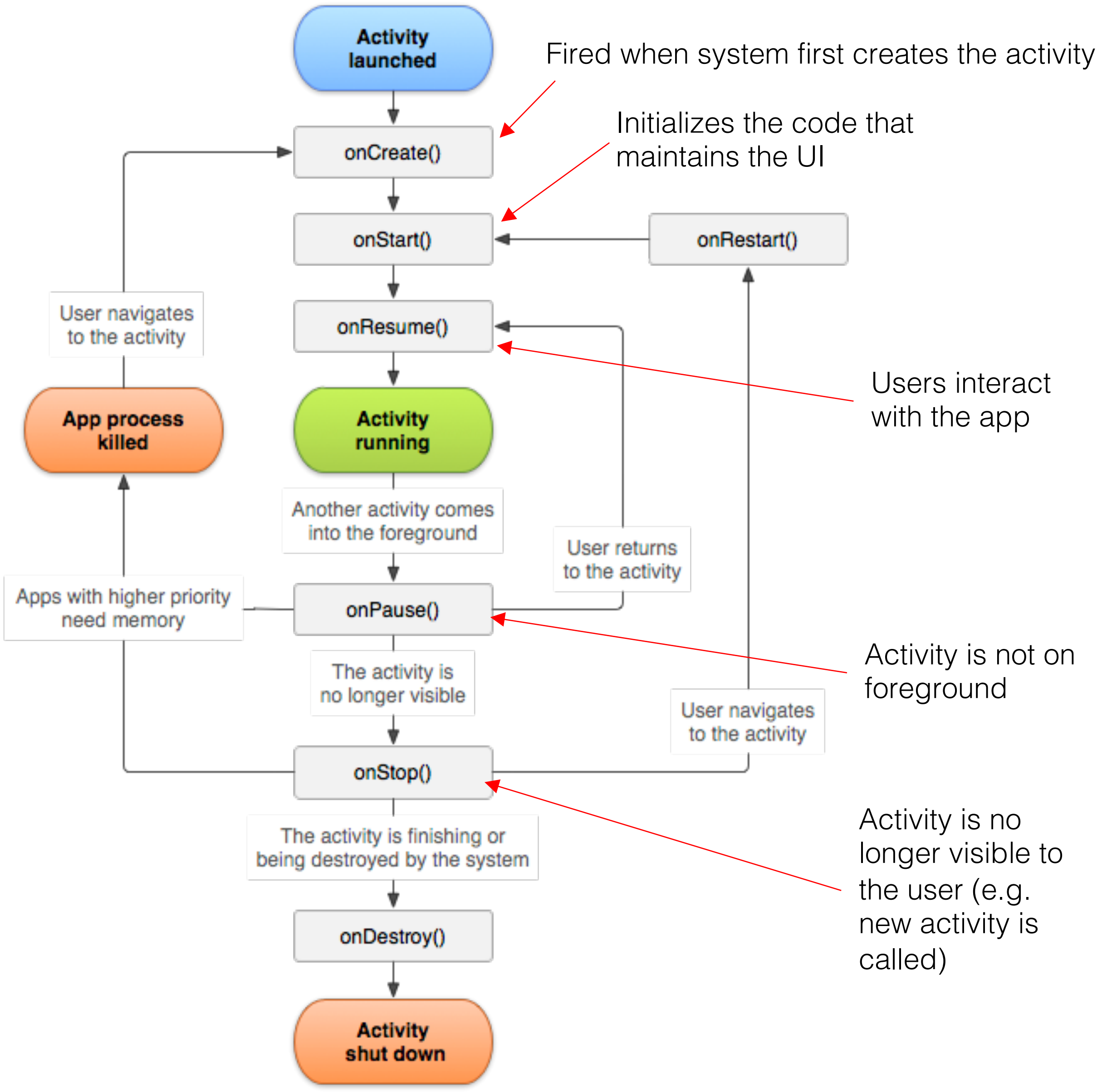
onCreate(Bundle) is where you initialize your activity. Most importantly, here you will usually call setContentView(int) with a layout resource defining your UI, and using findViewById(int) to retrieve the widgets in that UI that you need to interact with programmatically.

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

onPause() is where you deal with the user pausing active interaction with the activity. Any changes made by the user should at this point be committed (usually to the ContentProvider holding the data). In this state the activity is still visible on screen.

Activities

Activity **Lifecycle**


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down
Fired when system first creates the activity
Initializes the code that maintains the UI
Users interact with the app
Activity is not on foreground
Activity is no longer visible to the user (e.g. new activity is called)

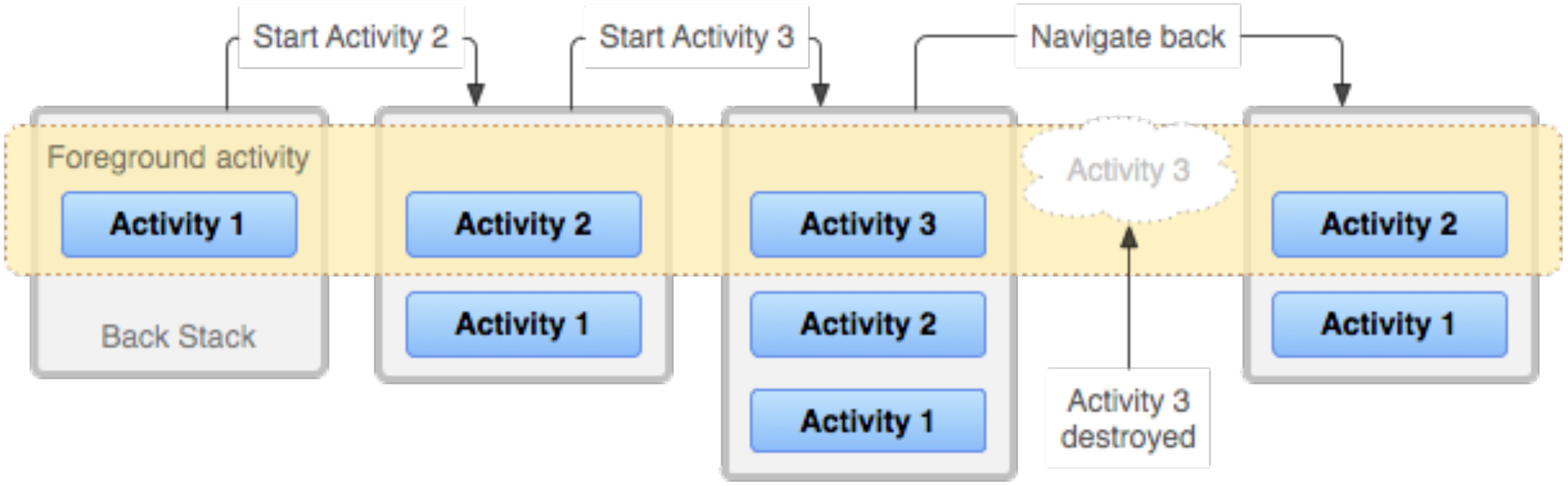
Activities

Activity **Lifecycle**

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast.makeText(getApplicationContext(),"onCreate!", Toast.LENGTH_LONG).show();
    }
    @Override
    protected void onStart() {
        super.onStart();
        Toast.makeText(getApplicationContext(),"onStart!", Toast.LENGTH_LONG).show();
    }
    @Override
    protected void onRestart(){
        super.onRestart();
        Toast.makeText(getApplicationContext(),"onRestart!", Toast.LENGTH_LONG).show();
    }
    @Override
    protected void onResume(){
        super.onResume();
        Toast.makeText(getApplicationContext(),"onResume!", Toast.LENGTH_LONG).show();
    } . . . }
```

Activities

Activities **Stack**



New Project

app > java > com.example.myfirstapp > MainActivity
This is the main activity. It's the entry point for your app. When you build and run your app, the system launches an instance of this Activity and loads its layout.

app > res > layout > activity_main.xml
This XML file defines the layout for the activity's user interface (UI). It contains a TextView element with the text "Hello, World!"

Example time - run on emulator

Apps adapt to different devices

> Android allows you to provide different resources for different devices. For example, you can create different layouts for different screen sizes. The system determines which layout to use based on the screen size of the current device.

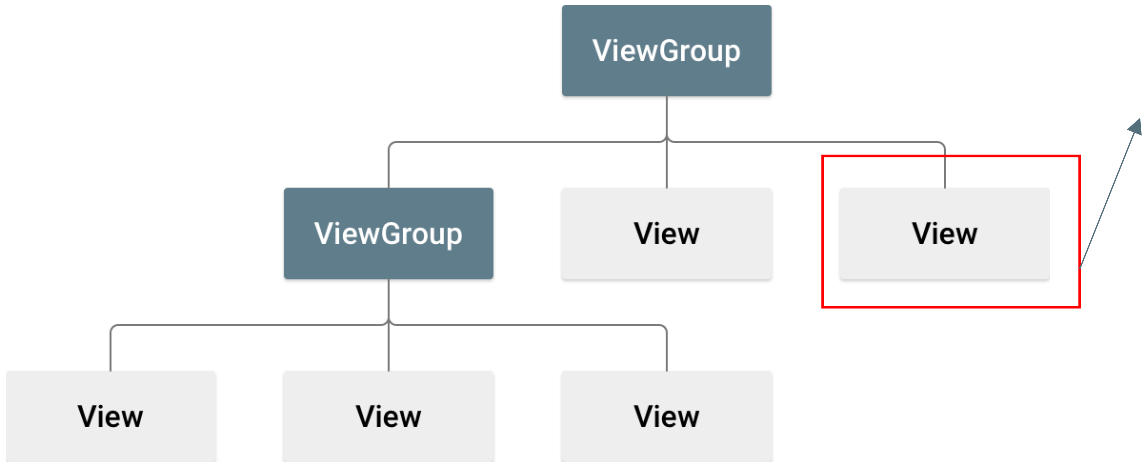
> If any of your app's features need specific hardware, such as a camera, you can query at runtime whether the device has that hardware or not, and then disable the corresponding features if it doesn't. You can specify that your app requires certain hardware so that Google Play won't allow the app to be installed on devices without them.

User Interface

A user interface is everything a user interacts with.
Android User interface (UI) is a hierarchy of layouts and widgets.

"*The layouts are* ***ViewGroups*** *objects, containers that control how their child views are positioned on the screen.* ***Widgets are View objects****, UI components such as buttons and text boxes.*"

View objects can be instantiated in code or defined in XML layout files.



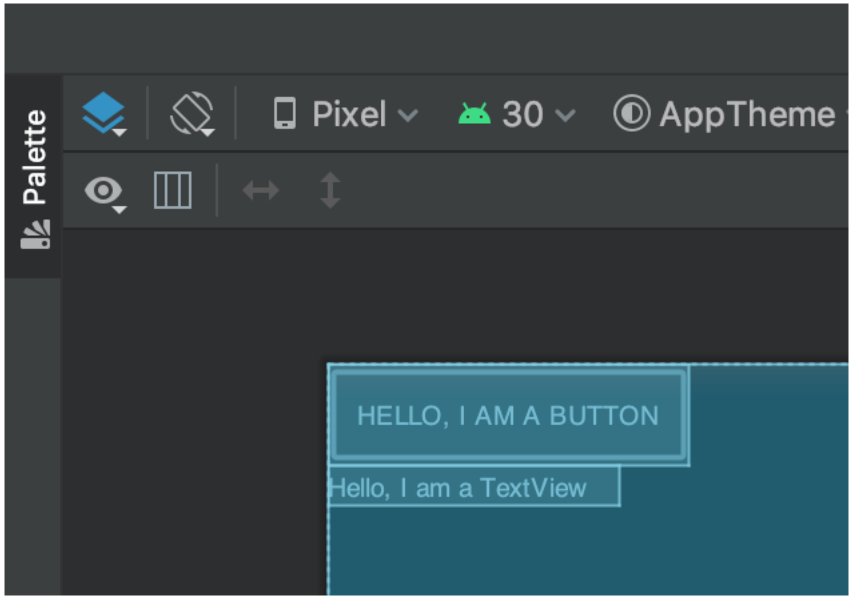
Objects that draws something on the screen (buttons, text boxes, …)

Layout

ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout.

a) LinearLayout arranges other views either horizontally in a single column or vertically in a single row.

```
<?xml version="1.0" encoding="utf-8"?>
   <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical" >

      <Button
         android:id="@+id/button"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Hello, I am a Button" />

      <TextView
         android:id="@+id/text"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Hello, I am a TextView" />

   </LinearLayout>
```



Australian National University

Layout

b) A **ConstraintLayout** is a **android.view.ViewGroup** which allows you to position and size widgets in a flexible way.

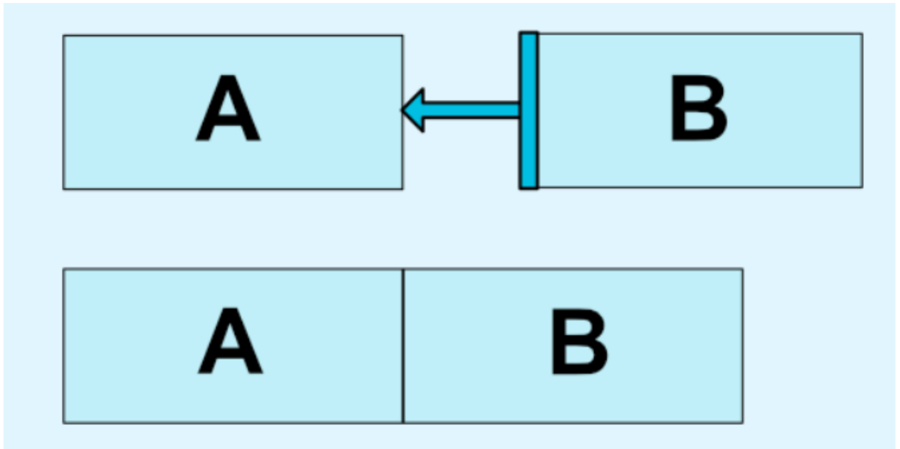
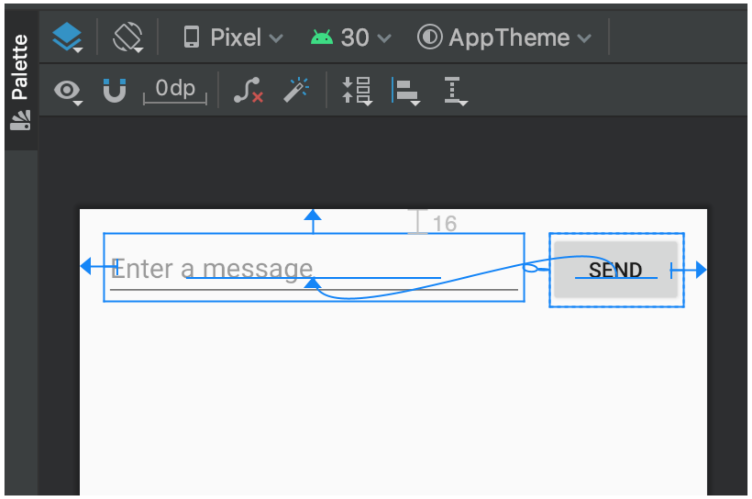


Fig. 1 - Relative Positioning Example



Layout

Two methods for declaration: Declare UI elements in XML and Instantiate layout elements at runtime.

Declare UI elements in XML.

- Well-defined XML vocabulary corresponding to widgets and layouts
- Separation between presentation of the app and controller
- Easy to maintain, provide different screen sizes and orientations, …
- Android Studio provides a drag-and-drop interface (generates the XML for you!)
-

  Even if you use this method, you can modify your layout at runtime if needed

Australian National University

Layout

Load the XML Resource

1) When you compile your app, each XML layout file is compiled into a View resource.

2) The layout resource should be loaded using the Activity.onCreate() callback.

3) setContentView() loads R.layout.layout_file_name (e.g. main_layout.xml).

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

All XML files should be placed in **res/layout/filename_layout.xml**

Layout

Every UI Element has an ID

@ indicates that this item should be parsed
and used as an ID resource.

+ indicates that this resource should be
compiled and added to R (R contains all resources ids).

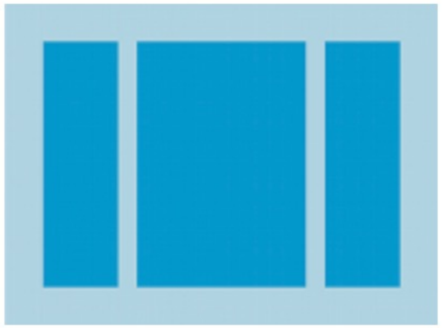
```
<Button
   android:id="@+id/mybutton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Hello, I am a Button" />

<TextView
   android:id="@+id/text"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Hello, I am a TextView" />
```

layout_width / height = [MATCH_PARENT | WRAP_CONTENT | EXACT_NUMBER]
Match_parent – match dimension to parent (do not consider padding)
Wrap content – just big enough to enclose its content (plus padding).

How to use these resources in Java code?
e.g. findViewById(R.id.mybutton);

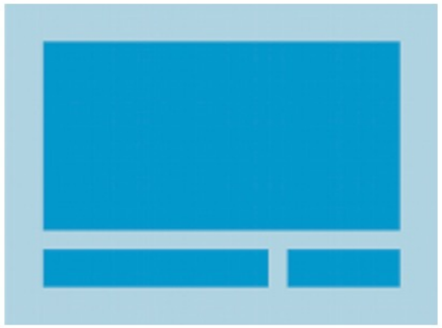
Common Layouts

Linear Layout



A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

Relative Layout



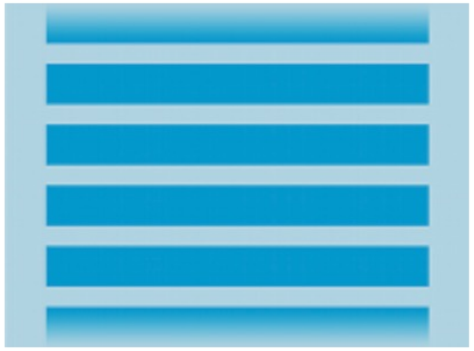
Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Web View

Displays web pages.

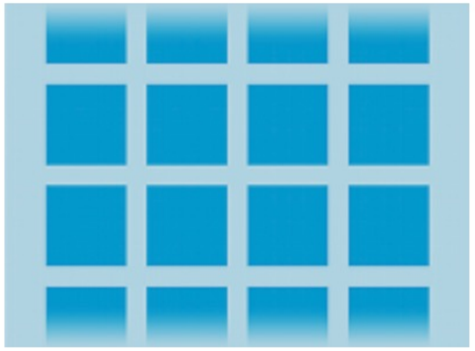
Dynamic Layouts (Adapter)

List View



Displays a scrolling single column list.

Grid View



Displays a scrolling grid of columns and rows.

When the content for your layout is dynamic or not pre-determined, you can use a layout that subclasses AdapterView to populate the layout with views at runtime.

Dynamic Layouts (Adapter)

Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView.

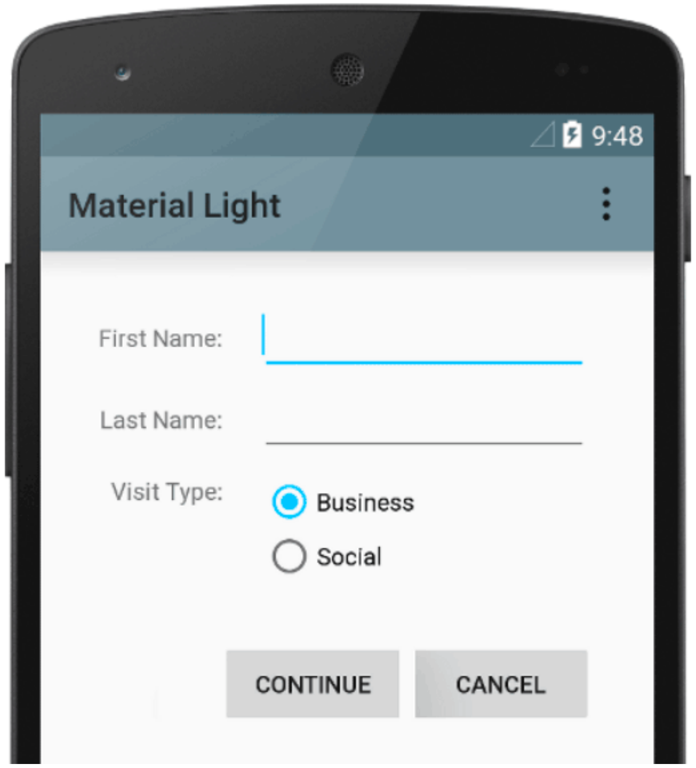
ArrayAdapter: Use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a TextView.

Example:

```
ArrayAdapter ad = new ArrayAdapter(this, android.R.layout.simple_list_item_1, arrItems);
```

… *and then setAdapter to the list/grid view.*

Input Controls



- Text
- Buttons
- Checkboxes
- Radio buttons
- Toggle buttons
- Spinners
- Pickers
- Tooltips

Example:
https://developer.android.com/guide/topics/ui/controls/checkbox

Australian National University

Input Events

Event Listeners

An event listener is an interface in the View class that contains a single callback method. **These methods will be** called by the Android framework when the View to which the listener has been registered is **triggered by user interaction with the item in the UI.**

onClick()
onLongClick()
onFocusChange()
onKey()
onTouch()
onCreateContextMenu()

Australian National University

Input Events

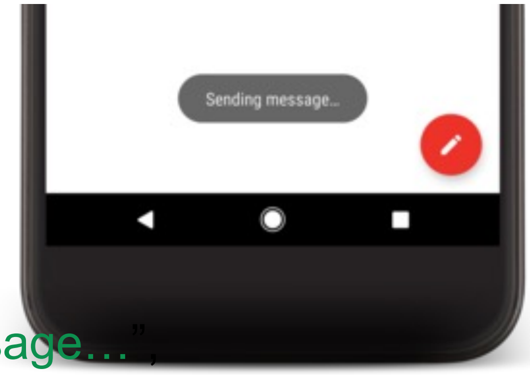
```
// Create an anonymous implementation of OnClickListener
private OnClickListener myListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.myId);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(myListener);
    ...
}
```

Toasts

A **toast** provides simple feedback about an operation in a **small popup**. It only fills the amount of space required for the message and the current activity remains visible and interactive.

Toasts **automatically disappear** after a timeout.
For example, clicking Send on an email triggers a "Sending message..." toast, as shown in the following screen capture:

```
Context context = getApplicationContext();
CharSequence text = "Sending Message…";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
//or
Toast.makeText(getApplicationContext(), "Sending Message…",
Toast.LENGTH_SHORT).show();
```



Australian National University

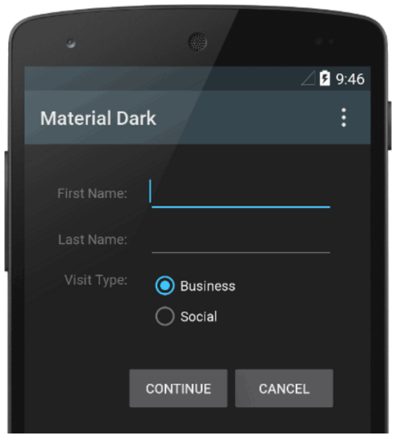
Themes and Style

Android allows you to separate the details of your app design from the UI structure and behavior, similar to stylesheets (CSS) in web design.

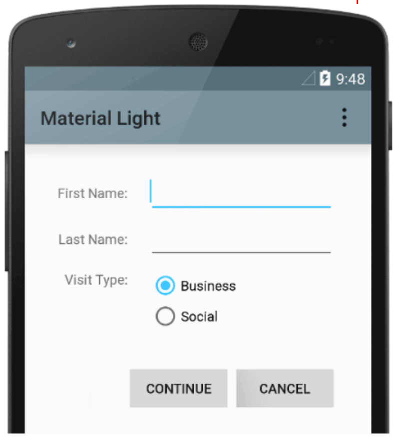
- A style can specify attributes such as font color, font size, background color, ...
- A theme is a type of style that's applied to an entire app, activity, or view hierarchy.
- Styles and themes are declared in a style resource file in res/values/, usually named styles.xml.

Themes and Style

```
<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>
```



Theme.AppCompat



Theme.AppCompat.Light.DarkActionBar