

Introduction to Mobile Programming

Android Programming

Chapter 2

Introduction to Activities

- ❖ The mobile-app experience differs from its desktop counterpart in that a user's interaction with the app doesn't always begin in the same place. Instead, the user journey often begins *non-deterministically*.
- ❖ The activity serves as the entry point for an app's interaction with the user.
- ❖ Most apps contain multiple screens, which means they comprise multiple activities.
- ❖ 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.

Configuring the manifest

- ❖ For your app to be able to use activities, you must declare the activities

```
<manifest ... >
  <application ... >
    <activity android:name=".ExampleActivity" />
    ...
  </application ... >
  ...
</manifest >
```

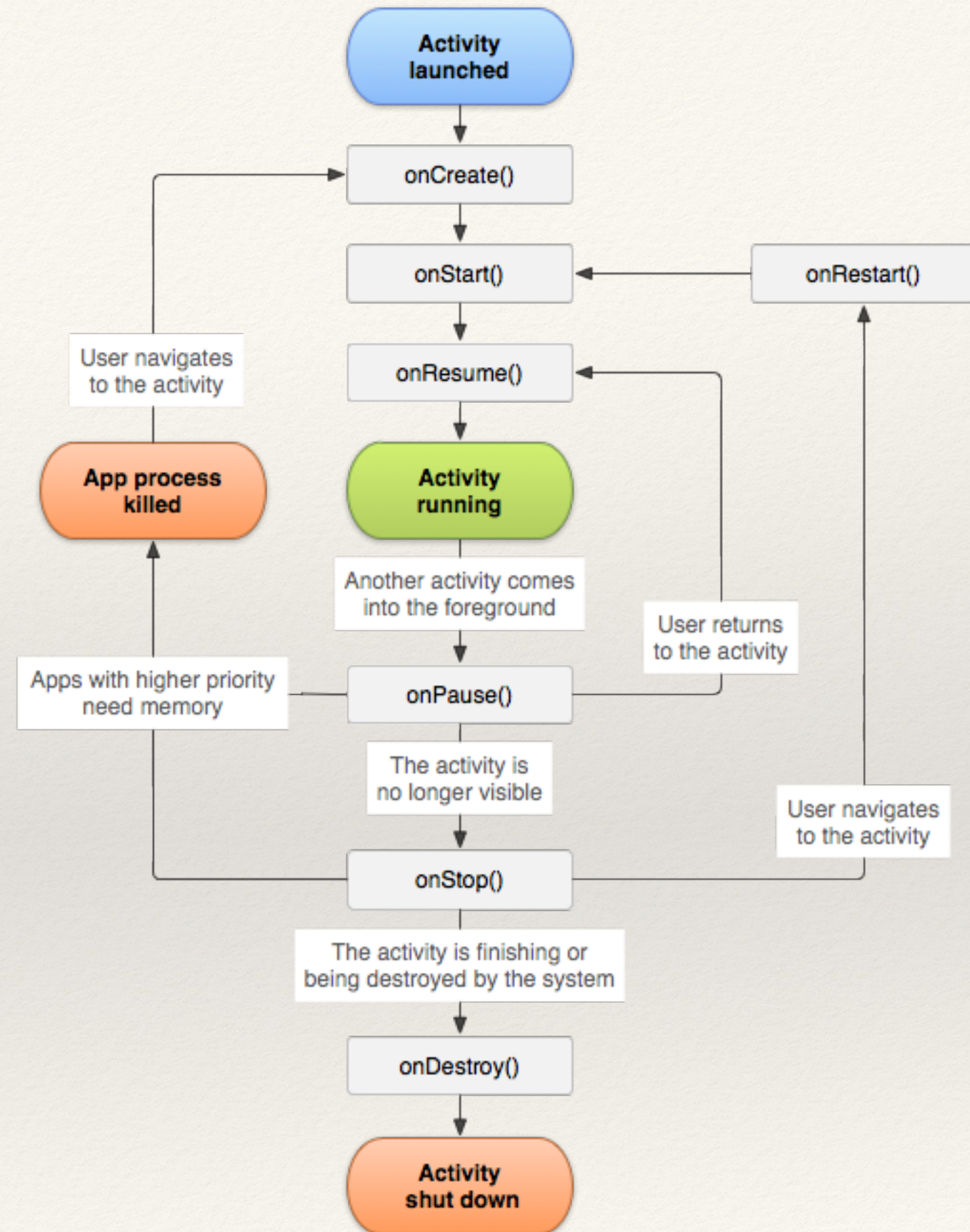
- ❖ **Intent filters** are a very powerful feature of the Android platform. They provide the ability to launch an activity based not only on an *explicit* request, but also an *implicit* one.

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
  <intent-filter>
    <action android:name="android.intent.action.SEND" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="text/plain" />
  </intent-filter>
</activity>
```


Activity-Lifecycle Concepts

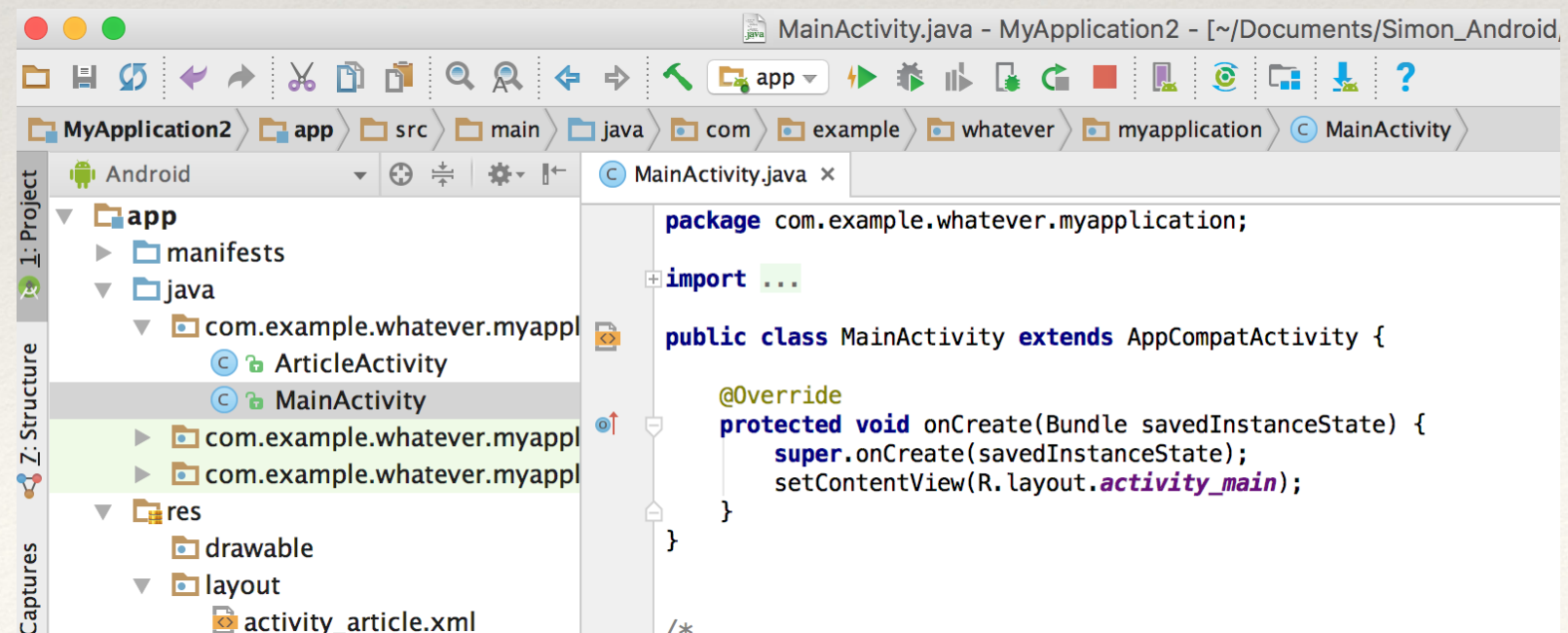
- Lifecycle Callbacks

- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onDestroy()



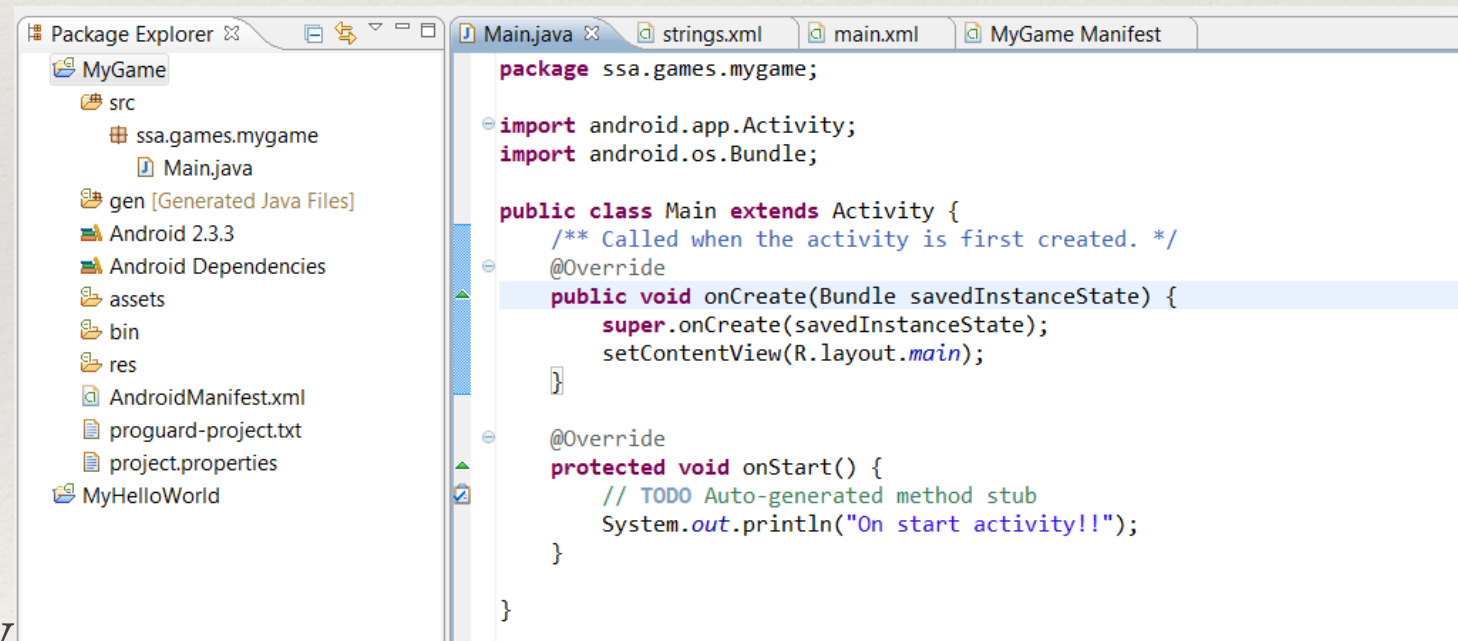
Activities - onCreate()

- This callback fires when the system first creates the activity.
 - Happen only once for the entire life of the activity
- ViewModel
- Bundle savedInstanceState



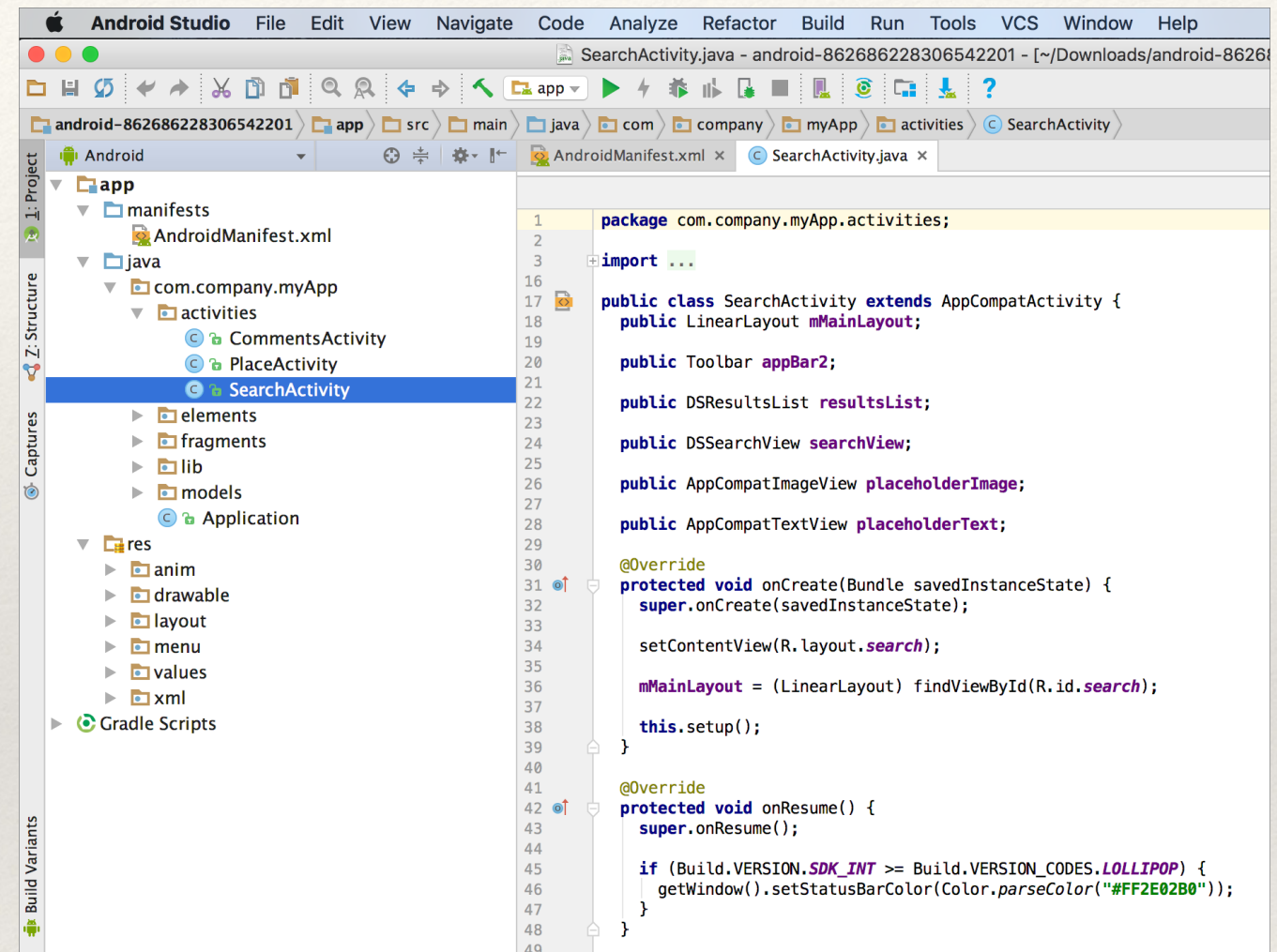
Activities - onStart()

- The app prepares for the activity to enter the foreground and become interactive.
- The `onStart()` method completes very quickly and, as with the Created state, the activity does not stay resident in the Started state.
- Once this callback finishes, the activity enters the *Resumed* state.



Activities - onResume()

- When the activity enters the Resumed state, it comes to the foreground, and then the system invokes the `onResume()` callback.
- This is the state in which the app interacts with the user. The app stays in this state until something happens to take focus away from the app.



Activities - onPause()

- It indicates that the activity is no longer in the foreground.
 - Some event interrupts app execution.
 - In Android 7.0 (API level 24) or higher, multiple apps run in multi-window mode. Because only one of the apps (windows) has focus at any time, the system pauses all of the other apps.
 - A new, semi-transparent activity (such as a dialog) opens. As long as the activity is still partially visible but not in focus, it remains paused.
- You can use the `onPause()` method to release system resources, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused and the user does not need them.



Activities - onStop()

- When your activity is no longer visible to the user, it has entered the *Stopped* state, and the system invokes the `onStop()` callback.
- The system may also call `onStop()` when the activity has finished running, and is about to be terminated.
- In the `onStop()` method, the app should release or adjust resources that are not needed while the app is not visible to the user.
- You should also use `onStop()` to perform relatively CPU-intensive shutdown operations.



Activities - onDestroy()

- The system invokes this callback either because:
 - the activity is finishing (due to the user completely dismissing the activity or due to `finish()` being called on the activity),
 - the system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode)



Activity state and ejection from memory

Likelihood of being killed	Process state	Activity state
Least	Foreground (having or about to get focus)	Created Started Resumed
More	Background (lost focus)	Paused
Most	Background (not visible)	Stopped
	Empty	Destroyed

Saving and Restoring Transient UI state

- ❖ When the activity is destroyed due to system constraints, you should preserve the user's transient UI state using a combination of `ViewModel`, `onSaveInstanceState()`, and / or local storage.
- ❖ User-initiated UI state dismissal and System-initiated UI state dismissal
- ❖ The user can completely dismiss an activity by:
 - ❖ pressing the back button
 - ❖ swiping the activity off of the Overview (Recents) screen
 - ❖ navigating up from the activity
 - ❖ killing the app from Settings screen
 - ❖ completing some sort of "finishing" activity (which is backed by `Activity.finish()`)

Options for preserving UI state

	ViewModel	Saved instance state	Persistent storage
Storage location	in memory	serialized to disk	on disk or network
Survives configuration change	Yes	Yes	Yes
Survives system-initiated process death	No	Yes	Yes
Survives user complete activity dismissal/onFinish()	No	No	Yes
Data limitations	complex objects are fine, but space is limited by available memory	only for primitive types and simple, small objects such as String	only limited by disk space or cost / time of retrieval from the network resource
Read/write time	quick (memory access only)	slow (requires serialization/deserialization and disk access)	slow (requires disk access or network transaction)


```

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    Button button;
    TextView textView;
    int counter;

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

        if (savedInstanceState != null) {
            String message = savedInstanceState.getString("message");
            Toast.makeText(this, message, Toast.LENGTH_LONG).show();

            counter = savedInstanceState.getInt("counter", 0);
        }

        button = findViewById(R.id.button);
        textView = findViewById(R.id.textView);
        textView.setText(String.valueOf(counter));

        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                counter = Integer.valueOf(textView.getText().toString()) + 1;
                textView.setText(String.valueOf(counter));
            }
        });
    }

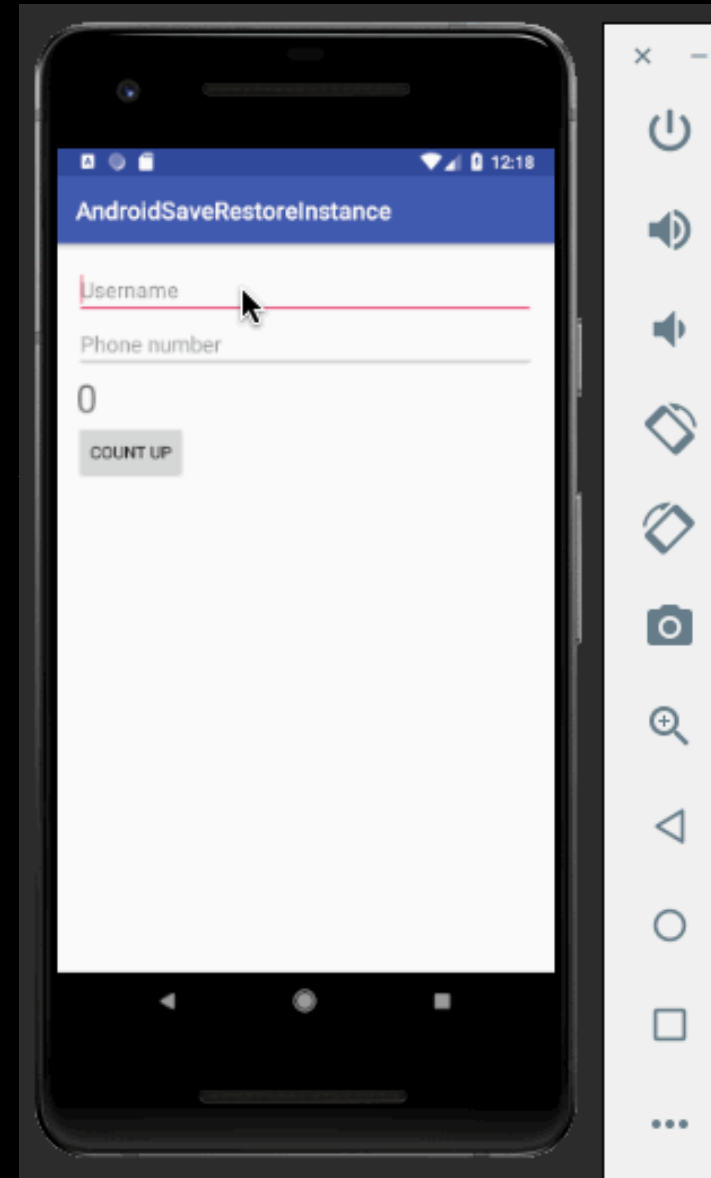
    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putString("message", "This is a saved message");
        outState.putInt("counter", counter);
    }
}

```

```

    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        Toast.makeText(getApplicationContext(), "onRestoreInstanceState",
            Toast.LENGTH_SHORT).show();
        counter = savedInstanceState.getInt("counter", 0);
    }
}

```



activity_main.xml

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Phone number"
    android:inputType="phone" />
```

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="0"
    android:textSize="32sp" />
```

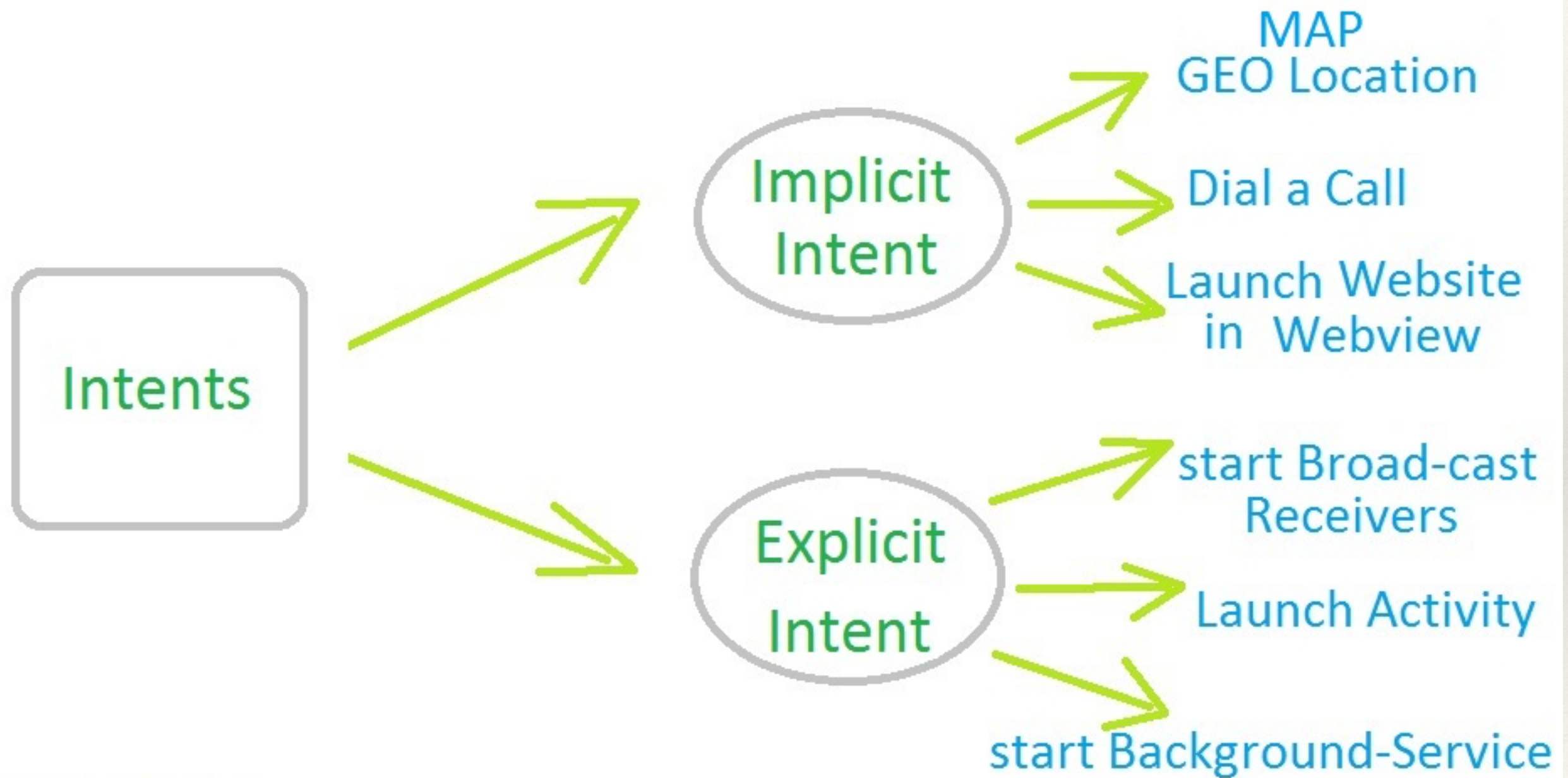
```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="COUNT UP" />
```

```
</LinearLayout>
```

Interacting with Other Apps

- To take the user from one activity to another, your app must use an `Intent` to define your app's "intent" to do something.
- When you pass an `Intent` to the system with a method such as `startActivity()`, the system uses the `Intent` to identify and start the appropriate app component.
- Using intents even allows your app to start an activity that is contained in a separate app.

Types of Intents - I



Types of Intents - II

Types of Intents

- Explicit intents
 - > Designate the target component by its name (the component name field is set by the class name)
 - > Since component names would generally not be known to developers of other applications, explicit intents are typically used for application-internal messages — such as an activity starting a subordinate service or launching a sister activity.
- Implicit intents
 - > Do not name a target (the component name field is blank).
 - > Implicit intents are often used to activate components in other applications.

Intents

Calling A Number

```
Uri number = Uri.parse("tel:5551234");  
Intent callIntent = new Intent(Intent.ACTION_DIAL, number);
```

Showing a Location on a Map

```
//  
/ Map point based on address  
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+California");  
/  
/ Or map point based on latitude/longitude  
// Uri location = Uri.parse("geo:37.422219,-122.08364?z=14"); // z param is zoom level  
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
```

Viewing a Webpage on a Browser

```
Uri webpage = Uri.parse("http://www.android.com");  
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);
```

Intents - II

Sending an E-mail

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);  
// The intent does not have a URI, so declare the "text/plain" MIME type  
emailIntent.setType(HTTP.PLAIN_TEXT_TYPE);  
emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[] {"jon@example.com"}); // recipients  
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Email subject");  
emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message text");  
emailIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse("content://path/to/email/attachment"));  
// You can also attach multiple items by passing an ArrayList of Uris
```

Creating and Adding and Calendar Event

```
Intent calendarIntent = new Intent(Intent.ACTION_INSERT, Events.CONTENT_URI);  
Calendar beginTime = Calendar.getInstance();  
beginTime.set(2012, 0, 19, 7, 30);  
Calendar endTime = Calendar.getInstance();  
endTime.set(2012, 0, 19, 10, 30);  
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, beginTime.getTimeInMillis());  
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_END_TIME, endTime.getTimeInMillis());  
calendarIntent.putExtra(Events.TITLE, "Ninja class");  
calendarIntent.putExtra(Events.EVENT_LOCATION, "Secret dojo");
```

Verify There is an App to Receive the Intent

- ❖ Although the Android platform guarantees that certain intents will resolve to one of the built-in apps (such as the Phone, Email, or Calendar app), you should always include a verification step before invoking an intent.
- ❖ If you invoke an intent and there is no app available on the device that can handle the intent, your app will crash.

- ```
PackageManager packageManager = getPackageManager\(\);
List<ResolveInfo> activities = packageManager.queryIntentActivities(intent,
 PackageManager.MATCH_DEFAULT_ONLY);
boolean isIntentSafe = activities.size() > 0;
```



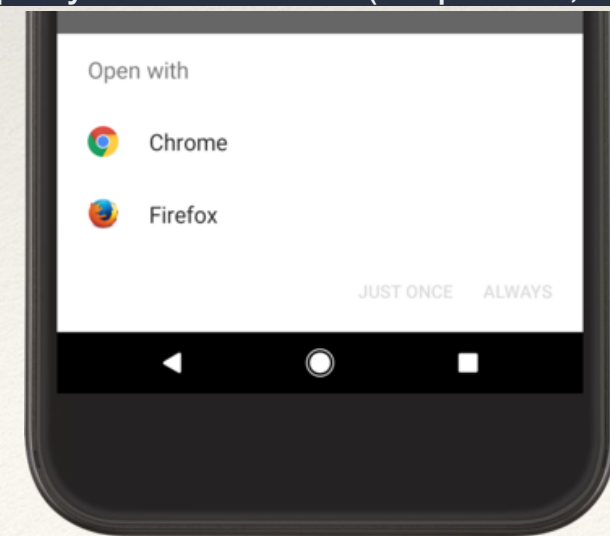
# Start an Activity with the Intent

- ❖ Once you have created your `Intent` and set the extra info, call `startActivity()` to send it to the system. If the system identifies more than one activity that can handle the intent, it displays a dialog (sometimes referred to as the "disambiguation dialog") for the user to select which app to use

```
• // Build the intent
• Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+California");
• Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);

• // Verify it resolves
• PackageManager packageManager = getPackageManager();
• List<ResolveInfo> activities = packageManager.queryIntentActivities(mapIntent, 0);
• boolean isIntentSafe = activities.size() > 0;

• // Start an activity if it's safe
if (isIntentSafe) {
 startActivity(mapIntent);
}
```





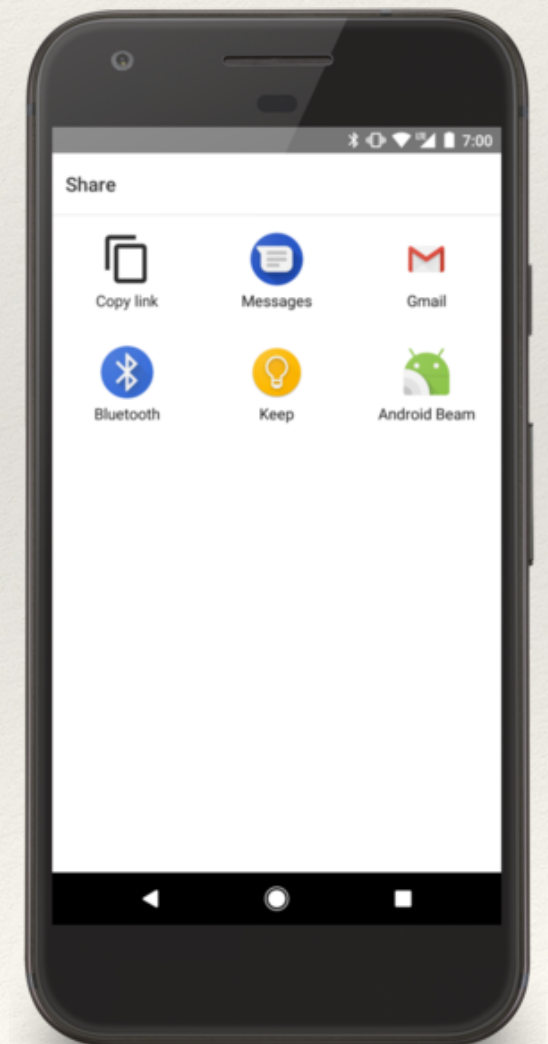
# Show an App Chooser

- ❖ The chooser dialog forces the user to select which app to use for the action every time (the user cannot select a default app for the action).

```
• Intent intent = new Intent(Intent.ACTION_SEND);
• ...

• // Always use string resources for UI text.
• // This says something like "Share this photo with"
• String title = getResources().getString(R.string.chooser_title);
• // Create intent to show chooser
• Intent chooser = Intent.createChooser(intent, title);

• // Verify the intent will resolve to at least one activity
if (intent.resolveActivity(getPackageManager()) != null) {
 startActivity(chooser);
}
```





# Getting a Result from an Activity

- ❖ Starting another activity doesn't have to be one-way. You can also start another activity and receive a result back. To receive a result, call `startActivityForResult()` (instead of `startActivity()`).

```
static final int PICK_CONTACT_REQUEST = 1; // The request code
...
private void pickContact() {
 Intent pickContactIntent = new Intent(Intent.ACTION_PICK, Uri.parse("content://contacts"));
 pickContactIntent.setType(Phone.CONTENT_TYPE); // Show user only contacts w/ phone numbers
 startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST);
}
```



---

# Receiving the Result

---

- ❖ When the user is done with the subsequent activity and returns, the system calls your activity's `onActivityResult()` method.
  - ❖ The request code you passed to `startActivityForResult()`.
  - ❖ A result code specified by the second activity. This is either `RESULT_OK` if the operation was successful or `RESULT_CANCELED` if the user backed out or the operation failed for some reason.
  - ❖ An `Intent` that carries the result data.



# Receiving The Result

@Override

```
protected void onActivityResult(int requestCode, int resultCode, Intent resultIntent) {
```

```
 // Check which request it is that we're responding to
```

```
 if (requestCode == PICK_CONTACT_REQUEST) {
```

```
 // Make sure the request was successful
```

```
 if (resultCode == RESULT_OK) {
```

```
 // Get the URI that points to the selected contact
```

```
 Uri contactUri = resultIntent.getData();
```

```
 // We only need the NUMBER column, because there will be only one row in the result
```

```
 String[] projection = {Phone.NUMBER};
```

```
 // Perform the query on the contact to get the NUMBER column
```

```
 // We don't need a selection or sort order (there's only one result for the given URI)
```

```
 // CAUTION: The query() method should be called from a separate thread to avoid blocking
```

```
 // your app's UI thread. (For simplicity of the sample, this code doesn't do that.)
```

```
 // Consider using CursorLoader to perform the query.
```

```
 Cursor cursor = getContentResolver()
```

```
 .query(contactUri, projection, null, null, null);
```

```
 cursor.moveToFirst();
```

```
 // Retrieve the phone number from the NUMBER column
```

```
 int column = cursor.getColumnIndex(Phone.NUMBER);
```

```
 String number = cursor.getString(column);
```

```
 // Do something with the phone number...
```

```
 }
 }
}
```



---

# Allowing Other Apps to Start Your Activity

---

```
<activity android:name="ShareActivity">
 <intent-filter>
 <action android:name="android.intent.action.SEND"/>
 <category android:name="android.intent.category.DEFAULT"/>
 <data android:mimeType="text/plain"/>
 <data android:mimeType="image/*"/>
 </intent-filter>
</activity>
```



# Handle the Intent in Your Activity

```
@Override
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);

 setContentView(R.layout.main);

 // Get the intent that started this activity
 Intent intent = getIntent();
 Uri data = intent.getData();

 // Figure out what to do based on the intent type
 if (intent.getType().indexOf("image/") != -1) {
 // Handle intents with image data ...
 } else if (intent.getType().equals("text/plain")) {
 // Handle intents with text ...
 }
}
```

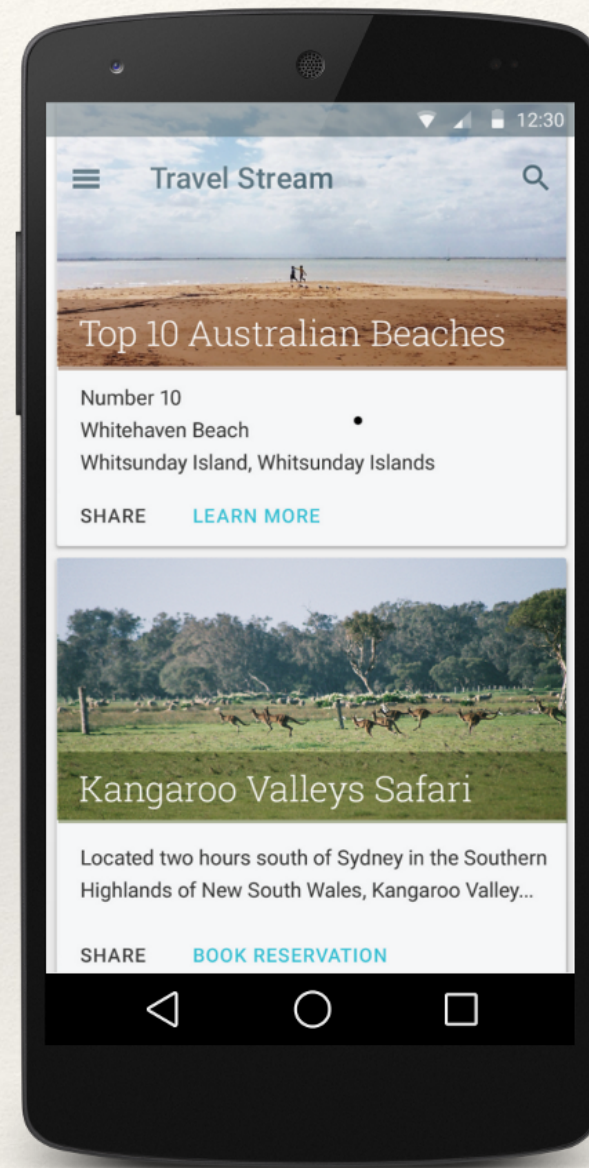
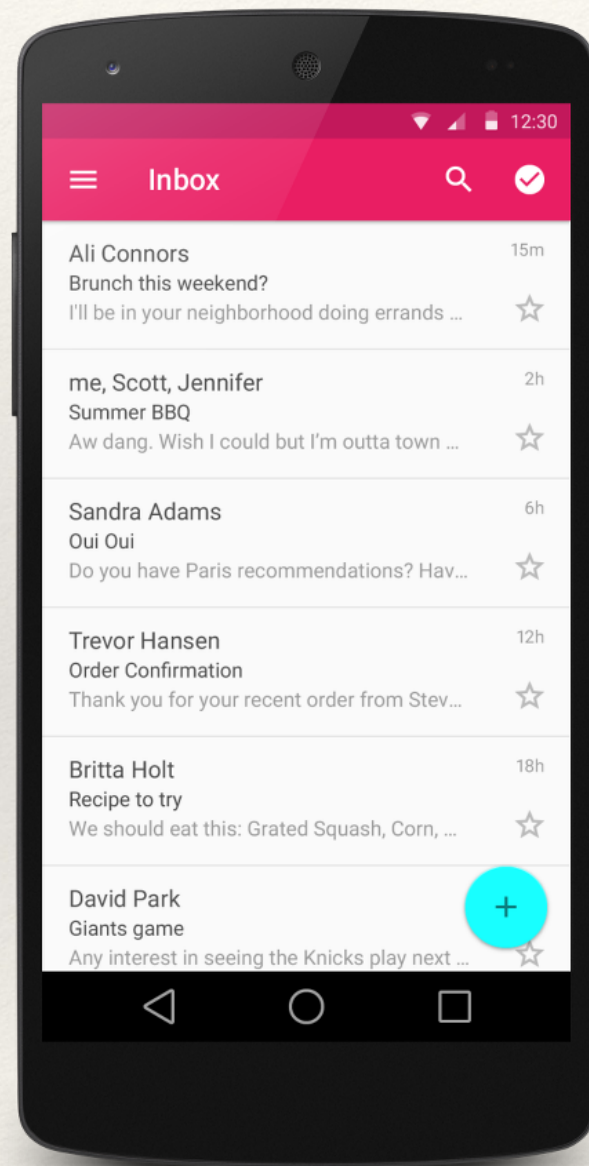
```
// Create intent to deliver some kind of result data
Intent result = new Intent("com.example.RESULT_ACTION", Uri.parse("content://result_uri"));
setResult(Activity.RESULT_OK, result);
finish();
```



**Note:** There's no need to check whether your activity was started with `startActivity()` or `startActivityForResult()`. Simply call `setResult()` if the intent that started your activity might expect a result. If the originating activity had called `startActivityForResult()`, then the system delivers it the result you supply to `setResult()`; otherwise, the result is ignored.



# Create a List with RecyclerView





---

# Create a List with RecyclerView

---

- ❖ The `RecyclerView` widget is a more advanced and flexible version of `ListView`.
- ❖ The `RecyclerView` fills itself with views provided by a *layout manager* that you provide. You can use one of our standard layout managers (such as `LinearLayoutManager` or `GridLayoutManager`), or implement your own.
- ❖ The views in the list are represented by *view holder* objects.
- ❖ The `RecyclerView` creates only as many view holders as are needed to display the on-screen portion of the dynamic content, plus a few extra.
- ❖ The view holder objects are managed by an *adapter*, which you create by extending `RecyclerView.Adapter`.



# RecyclerView

```
dependencies {
 implementation 'com.android.support:recyclerview-v7:28.0.0'
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<!-- A RecyclerView with some commonly used attributes -->
<android.support.v7.widget.RecyclerView
 android:id="@+id/my_recycler_view"
 android:scrollbars="vertical"
 android:layout_width="match_parent"
 android:layout_height="match_parent"/>
```

```
public class MyActivity extends Activity {
 private RecyclerView recyclerView;
 private RecyclerView.Adapter mAdapter;
 private RecyclerView.LayoutManager layoutManager;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.my_activity);
 recyclerView = (RecyclerView) findViewById(R.id.my_recycler_view);

 // use this setting to improve performance if you know that changes
 // in content do not change the layout size of the RecyclerView
 recyclerView.setHasFixedSize(true);

 // use a linear layout manager
 layoutManager = new LinearLayoutManager(this);
 recyclerView.setLayoutManager(layoutManager);

 // specify an adapter (see also next example)
 mAdapter = new MyAdapter(myDataset);
 recyclerView.setAdapter(mAdapter);
 }
 // ...
}
```



```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.MyViewHolder> {
 private String[] mDataset;
```

```
 // Provide a reference to the views for each data item
 // Complex data items may need more than one view per item, and
 // you provide access to all the views for a data item in a view holder
 public static class MyViewHolder extends RecyclerView.ViewHolder {
 // each data item is just a string in this case
 public TextView textView;
 public MyViewHolder(TextView v) {
 super(v);
 textView = v;
 }
 }
}
```

```
 // Provide a suitable constructor (depends on the kind of dataset)
 public MyAdapter(String[] myDataset) {
 mDataset = myDataset;
 }
```

```
 // Create new views (invoked by the layout manager)
 @Override
 public MyAdapter.MyViewHolder onCreateViewHolder(ViewGroup parent,
 int viewType) {
 // create a new view
 TextView v = (TextView) LayoutInflater.from(parent.getContext())
 .inflate(R.layout.my_text_view, parent, false);
 ...
 MyViewHolder vh = new MyViewHolder(v);
 return vh;
 }
```

```
 // Replace the contents of a view (invoked by the layout manager)
 @Override
 public void onBindViewHolder(MyViewHolder holder, int position) {
 // - get element from your dataset at this position
 // - replace the contents of the view with that element
 holder.textView.setText(mDataset[position]);

 }
```

```
 // Return the size of your dataset (invoked by the layout manager)
 @Override
 public int getItemCount() {
 return mDataset.length;
 }
```