

Android

Ben Gavan

October 12, 2019

Contents

1	Fragments	5
1.0.1	isAdded()	5
2	Dialogs	5
2.1	Creating a Dialog View	5
2.1.1	XML	6
2.1.2	The Class	6
2.2	Setting a Target Fragment	8
2.3	Sending data back to the Target Fragment from the Dialog . . .	8
2.3.1	Receiving the Data from the intent	8
3	The Toolbar	9
3.0.1	History	9
3.0.2	Supported by	9
3.1	Menus	10
3.1.1	Defining a menu in XML	10
3.1.2	Defining an item	10
3.1.3	Creating the Menu	11
3.1.4	Responding to Menu Selection	12
3.1.5	Reload/update the menu	12
3.2	Subtitle	12
3.3	Hierarchical Navigation	13
3.3.1	How Hierarchical Navigation works	13
4	AppCompat Library	13
4.1	Requirements	13
5	SQLite Database	13
5.1	Defining a Schema	13
5.2	Building the Initial Database	13
5.3	Opening an SQLiteDatabase	14
5.4	Writing to the Database	14
5.4.1	ContentValues	14

5.4.2	Inserting rows	15
5.4.3	Updating Rows	15
5.5	Reading from the Database	15
5.5.1	Retrieving a Cursor	15
5.5.2	Using a Cursor	16
5.6	Deleting Rows	17
6	Implicit Intents	17
6.1	The Parts of an Implicit Intent	17
6.2	Advertising an Activity to Accept Implicit Intents	18
6.3	Sending Text	18
6.3.1	With option of send	18
6.4	Requesting Android for a Contact	18
6.4.1	The Request of Data	18
6.4.2	The Receival of Data	19
6.4.3	Checking if the Device has a Contacts App	19
6.5	Taking Pictures with intents	19
6.5.1	Declaring the Camera Feature	20
7	File Storage	20
7.1	FileProvider	21
7.1.1	Exposing/Telling the <i>FileProvider</i> what files it is exposing	21
7.1.2	Hooking up the paths description to the <i>FileProvider</i> within <i>AndroidManifest.xml</i>	22
7.1.3	Revoking File write <i>FileProvider</i> permissions	22
8	Bitmaps	22
8.1	Scaling Bitmaps	22
8.1.1	Very conservative scaling	23
9	Strings	23
9.1	Plurals	23
9.2	Percentages	24
9.3	Formatted Strings	24
9.3.1	Strings with values embedded	24
10	Intents	24
10.1	Starting n activity in a new task	24
11	Two-Pane Master-Detail	24
11.1	Alias Resource	24
11.2	Determining Device size	25

12 RecyclerView	25
12.1 Item cell padding	25
12.2 Swipe to delete	26
12.2.1 Creating <i>ItemTouchHelper.SimpleCallback</i>	26
12.3 Press and hold to change order	27
12.4 Rounded Corners	27
13 Localization	28
14 Accessibility	28
14.1 TalkBack	28
15 Styles and Themes	28
15.1 Styles	28
15.2 Style Inheritance	28
15.3 Themes	28
15.3.1 android:windowBackground	29
15.3.2 Overriding themes	29
15.3.3 Accessing Theme Attributes	29
16 Assets	29
16.1 Why Assets over Dependencies	29
16.2 Creating an Assets Folder	29
16.3 Accessing Assets	30
16.3.1 Getting an <i>AssetsManager</i>	30
16.3.2 Getting assets files names	30
16.4 SoundPool	30
16.4.1 Creating a SoundPool	30
16.4.2 Loading Sounds	31
16.4.3 Playing Sounds	31
17 XML Drawables	31
17.1 What is a Drawable	31
17.2 State List Drawables	31
17.3 shape drawables	32
17.4 layer list drawables	32
18 Custom Views	33
18.1 Custom Simple View	33
18.1.1 Background Color	33
18.2 Handling Touch Events	33
18.2.1 <i>MotionEvent</i>	34
18.3 Rendering inside <i>onDraw(Canvas)</i>	34
19 Animation	35
19.1 Interpolation	35
19.1.1 Setting the interpolator	35

20 9-Patch Images	35
21 AsyncTask	35
21.1 <i>AsyncTasks.cancel(boolean)</i>	35
21.1.1 <i>.cancel(true)</i>	35
21.1.2 <i>.cancel(false)</i>	35
21.2 message queue	35
21.3 Looper	35
21.4 Handler	36
21.5 Background Thread	36
21.5.1 Assembling a Background Thread	36
21.6 <i>ConcurrentHashMap</i>	36
22 HTTP	36
22.1 Make get requests	36
22.2 Connecting to local sever from emulator	36
22.3 building a Url String	37
22.4 Parsing JSON	37
23 LruCache	37
23.1 Creating a new Cache	37
23.2 Retrieval from the Cache	37
23.3 Adding item to cache	37
24 SearchView	38
25 SharedPreferences	38
26 Background Services	38
26.1 <i>IntentService</i>	38
26.1.1 Basic Skeletal example	39
26.2 <i>commands</i>	39
26.3 Safe background networking	39
26.3.1 Checking for background networking availability	39
26.4 <i>AlarmManager</i>	39
27 Notifications	40
27.1 Creating a notification	40
28 Broadcast Intents	40
28.1 Regular <i>Intents</i> vs <i>Broadcast Intents</i>	40
28.2 Receiving a system Broadcast	40
28.2.1 Waking Up on Boot	40
28.2.2 Creating and Registering a standalone receiver	41
28.3 Sending Broadcast Intents	41
28.4 <i>Dynamic Broadcast Receiver</i>	41
28.4.1 Register	41

28.4.2	Un-Register	42
28.5	<i>IntentFilter</i>	42
28.6	For longer running tasks	42
29	Location	42
29.1	Requirements	42
29.1.1	Checking Play Services are Available	43
29.1.2	Location Permissions	43
29.1.3	<i>GoogleApiClient</i>	43
29.2	Getting a Location Fix	44
29.2.1	Checking You have location permissions	45
29.2.2	Requesting location permissions	45
29.3	Requesting Location	46
30	Material Design	46
31	StrictMode	46
32	Unit Testing	47
32.1	Creating a Test Class	47
32.1.1	<i>androidTest</i> vs <i>test</i>	47
32.2	Setting Up the Test	47
32.2.1	Using Mocked Dependencies	47
32.3	Writing Tests	47
33	Refactoring Techniques and Tools	47
33.1	Extracting a method with Android Studio	47
34	Building the APK	48
34.1	Separate APK for Screen size	48
34.1.1	Exception	48

1 Fragments

1.0.1 isAdded()

Checks whether the fragment has been attached to an activity, therefore *getActivity()* will not be *null*.

2 Dialogs

2.1 Creating a Dialog View

A dialog is a type of fragment.

2.1.1 XML

To create the Dialog View, it is just like creating any other view. Create a normal layout xml file and create the layout that is required for the dialog.

Example: DatePicker :

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <DatePicker xmlns:android="http://schemas.android.com/apk/res/
  android"
3   android:id="@+id/dialog_date_picker"
4   android:layout_width="wrap_content"
5   android:layout_height="wrap_content"
6   android:calendarViewShown="false">
7 </DatePicker>
```

2.1.2 The Class

A Dialog Fragment extends the class *DialogFragment*

```
1 public class DatePickerFragment extends DialogFragment {
2   ...
3 }
```

Example: DatePicker :

```
1
2 import android.app.Activity;
3 import android.app.Dialog;
4 import android.content.DialogInterface;
5 import android.content.Intent;
6 import android.os.Bundle;
7 import android.support.annotation.NonNull;
8 import android.support.v4.app.DialogFragment;
9 import android.support.v7.app.AlertDialog;
10 import android.view.LayoutInflater;
11 import android.view.View;
12 import android.widget.DatePicker;
13
14 import java.util.Calendar;
15 import java.util.Date;
16 import java.util.GregorianCalendar;
17
18 /**
19  * Created by ben on 26/10/2017.
20  */
21
22 public class DatePickerFragment extends DialogFragment {
23
24   public static final String EXTRA_DATE = "com.bgsoftwarestudios.
     criminalintent.date";
25
26   private static final String ARG_DATE = "date";
27 }
```

```

28 private DatePicker mDatePicker;
29
30 public static DatePickerFragment newInstance(Date date) {
31     Bundle args = new Bundle();
32     args.putSerializable(ARG_DATE, date);
33
34     DatePickerFragment fragment = new DatePickerFragment();
35     fragment.setArguments(args);
36     return fragment;
37 }
38
39 @NonNull
40 @Override
41 public Dialog onCreateDialog(Bundle savedInstanceState) {
42     Date date = (Date) getArguments().getSerializable(ARG_DATE);
43
44     Calendar calendar = Calendar.getInstance();
45     calendar.setTime(date);
46     int year = calendar.get(Calendar.YEAR);
47     int month = calendar.get(Calendar.MONTH);
48     int day = calendar.get(Calendar.DAY_OF_MONTH);
49
50     View view = LayoutInflater.from(getActivity()).inflate(R.layout
        .dialog_date, null);
51
52     mDatePicker = (DatePicker) view.findViewById(R.id.
        dialog_date_picker);
53     mDatePicker.init(year, month, day, null);
54
55     return new AlertDialog.Builder(getActivity())
56         .setView(view)
57         .setTitle(R.string.date_picker_title)
58         .setPositiveButton(android.R.string.ok, new
        DialogInterface.OnClickListener() {
59             @Override
60             public void onClick(DialogInterface dialogInterface,
        int i) {
61                 int year = mDatePicker.getYear();
62                 int month = mDatePicker.getMonth();
63                 int day = mDatePicker.getDayOfMonth();
64                 Date date = new GregorianCalendar(year, month, day)
        .getTime();
65                 setResult(Activity.RESULT_OK, date);
66             }
67         })
68         .create();
69 }
70
71 private void sendResult(int resultCode, Date date) {
72     if (getTargetFragment() == null) {
73         return;
74     }
75
76     Intent intent = new Intent();
77     intent.putExtra(EXTRA_DATE, date);
78

```

```

79         this.getTargetFragment().onActivityResult(this.
           getTargetRequestCode(), resultCode, intent);
80     }
81 }

```

2.2 Setting a Target Fragment

When displaying a dialog view from a fragment, we need to create a relationship between them so we can send data back from the dialog to the fragment.

We need to pass a reference to the dialog of the fragment, as well as a request code to identify the payload when it is sent back/ so the fragment can 'listen' out for it.

We do this by setting the target fragment on the dialog object:

```

1 dialog.setTargetFragment(FragmentClass.this, REQUEST_CODE);

```

2.3 Sending data back to the Target Fragment from the Dialog

We should also check that the target fragment has been set before we do anything

First, we need to get a reference to the target fragment (set by the fragment requesting the display of the dialog via using setTargetFragment on the dialog). We then call 'onActivityResult' on the target fragment.

So if we want to do something in the fragment i.e. get the data back, we have to override this method in the target fragment.

The data we pass back from the dialog is contained within an intent by putExtra.

```

1 private void sendResult(int resultCode, Date date) {
2     if (getTargetFragment() == null) {
3         return;
4     }
5
6     Intent intent = new Intent();
7     intent.putExtra(EXTRA_DATE, date);
8
9     this.getTargetFragment().onActivityResult(this.
       getTargetRequestCode(), resultCode, intent);
10 }

```

2.3.1 Receiving the Data from the intent

```

1 @Override
2 public void onActivityResult(int requestCode, int resultCode,
   Intent data) {
3     if (resultCode != Activity.RESULT_OK) {
4         return;
5     }
6
7     if (requestCode == REQUEST_DATE) {

```



```

8      Date date = (Date) data.getSerializableExtra(DatePickerFragment
          .EXTRA_DATE);
9      mCrime.setDate(date);
10     mDateButton.setText(mCrime.getDate().toString());
11 }
12 }

```

We override the 'onActivityResult' within the target fragment we are sending data back to.

First we check what the result code is (what button the user pressed on the dialog)

```

1 if (resultCode != Activity.RESULT_OK) {
2     return;
3 }

```

We then check what the request code is (which was set by the fragment creating the dialog) so we know that we are responding to the correct result (A fragment can display and react to multiple dialogs).

After this, we get the data sent back in the form of an extra from the dialog inside an intent by 'getSerializableExtra(...)'.

In this case, we cast this data back to a date so it can be used.

```

1 if (requestCode == REQUEST_DATE) {
2     Date date = (Date) data.getSerializableExtra(DatePickerFragment.
        EXTRA_DATE);
3     mCrime.setDate(date);
4     mDateButton.setText(mCrime.getDate().toString());
5 }

```

3 The Toolbar

The Toolbar provides additional mechanisms for navigation, and also provides design consistency and branding.

3.0.1 History

The toolbar component was added to android 5.0 (Lollipop).

Prior to this, the action bar was the recommended component for navigation and actions within an app.

The toolbar and action bar are very similar.

The toolbar builds on top of the action bar .

It has a tweaked UI

It's more flexible in the ways you can use it.

3.0.2 Supported by

Since the toolbar has been added to the AppCompatActivity library, it is available back to API 9 (Android 2.3)

3.1 Menus

The top-right portion of the toolbar is reserved for the toolbar's menu. The menu consists of action items (sometimes referred to as menu items). These can perform an action on the current screen or on the app as a whole.

3.1.1 Defining a menu in XML

Need to create an XML description of a menu, just like how you have to for layouts, with the resource file inside the res/menu directory.

To create a new menu resource file:

1. Right-click on the res directory
2. Select New → Android resource file
3. Change the Resource type to Menu
4. Name the resource (normally 'fragment....' - the same naming convention as layout files)
5. Click OK

In this file, the XML should be:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto">
4 </menu>
```

3.1.2 Defining an item

```
1 <item
2   android:id="@+id/new_crime"
3   android:icon="@android:drawable/ic_menu_add"
4   android:title="@string/new_crime"
5   app:showAsAction="ifRoom|withText"/>
```

The line

```
1 app:showAsAction="ifRoom|withText "
```

makes the item be displayed inline/on the toolbar (where the menu icon should be) instead of having the item as a drop down item below the toolbar/menu button.

The showAsAction attribute refers to whether the item will appear in the toolbar itself or in the overflow menu.

In this case "ifRoom|withText" will make the items icon and text appear in the toolbar if there is room.

If there is room for the icon but not the text, then only the icon will be visible.

If there is no room for either, the item will be relegated to the overflow menu.

If there are items in the overflow menu, the three dots will appear and when these are pressed, the overflow menu will be shown below.

Multiple menu items can be displayed as Actions on the Toolbar.

Possible values for `showAsAction`

- `always`
 - not recommended
 - Better to use `ifRoom` and let the OS decide.
- `ifRoom`
 - Only displays the item as an Action if there is room
- `never`
 - never displayed as an action
 - will always appear in the overflow menu
 - so good for items that are not used very often - its good practice to avoid having too many items on the toolbar to help the screen keep decluttered

The AppCompatActivity library defines its own custom `showAsAction` attribute and does not look for the native `showAsAction` attribute.

3.1.3 Creating the Menu

Override the function `onCreateOptionsMenu(...)` inside the Activity/Fragment. To actually create/inflate the menu: (Inside the Fragment:)

```
1 @Override
2 public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
3     super.onCreateOptionsMenu(menu, inflater);
4     inflater.inflate(R.menu.fragment_crime_list, menu);
5 }
```

This populates the menu with the items defined in the `menu/fragment_crime_list.xml` file.

The super call is only convention since the superclass, `Fragment`, does nothing. (Good to do so the superclass functionality is still applied - can now change the superclass and will still work if we do something in that implementation of this function).

We then need to call `setHasOptionsMenu(boolean hasMenu)` to tell the `FragmentManager` that this fragment has a menu and should receive a call to `onCreateOptionsMenu(...)`.

Inside the Fragment:

```

1 @Override
2 public void onCreate(@Nullable Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4     setHasOptionsMenu(true);
5 }

```

3.1.4 Responding to Menu Selection

Override `onOptionsItemSelected(...)` in the fragment that you have called `'setHasOptionsMenu(true)'`.

The `MenuItem.getItemId()` corresponds to the id of the `<item>` which you set in the xml file for the menu.

This means that we can perform a switch case for each possible id in the menu. Include a default case to let the super implementation to handle the section of any item that you have not declared.

You should return true you you have handled the item section and that no further processing is necessary.

```

1 @Override
2 public boolean onOptionsItemSelected(MenuItem item) {
3     switch (item.getItemId()) {
4         case R.id.new_crime:
5             .... Do some logic here ....
6             return true; // Return true to say that the selection has
7                           been handled.
8         default:
9             return super.onOptionsItemSelected(item);
10    }

```

3.1.5 Reload/update the menu

```

1 getActivity().invalidateOptionsMenu();

```

This will cause the menu to be redrawn/reloaded (just like if the device is rotated).

3.2 Subtitle

```

1 private void setSubtitle(String subtitle) {
2     AppCompatActivity activity = (AppCompatActivity) getActivity();
3     activity.getSupportActionBar().setSubtitle(subtitle);
4 }

```

- Get the current activity
(We are using AppCompatActivity for backwards compatibility)
- Get the Toolbar from that activity via `getSupportActionBar()`
Still called/referred to as an Action Bar due to legacy reasons.
- Set the subtitle of that the toolbar we just received.

3.3 Hierarchical Navigation

Add `parentActivityName` to the activity the the manifests so when you press the back arrow on the toolbar, it will go back to the activity you stated.

```
1 <activity
2   android:name=".CrimePagerActivity"
3   android:parentActivityName=".CrimeListActivity"/>
```

3.3.1 How Hierarchical Navigation works

Page 261

4 AppCompatActivity Library

4.1 Requirements

The AppCompatActivity requires that you:

- add the AppCompatActivity dependency
- use one of the AppCompatActivity themes
- ensure that all activities are a subclass of AppCompatActivity

5 SQLite Database

5.1 Defining a Schema

```
1 public class CrimeDbSchema {
2
3     public static final class CrimeTable {
4         public static final String NAME = "crimes";
5
6         public static final class Cols {
7             public static final String UUID = "uuid";
8             public static final String TITLE = "title";
9             public static final String DATA = "date";
10            public static final String SOLVED = "solved";
11        }
12    }
13 }
```

5.2 Building the Initial Database

Always need to follow a few basic steps:

- Check to see whether the database already exists.
- If it does not, create it and create the tables and initial data it needs.
- If it does, open it and see what version of the schema it has.

- If it is an old version, upgrade it to a newer version.

SQLiteOpenHelper can be used to handle all of this.

5.3 Opening an SQLiteDatabase

By extending SQLiteOpenHelper, we give control over to it to do the heavy lifting in opening the database.

```

1 public class CrimeBaseHelper extends SQLiteOpenHelper {
2
3     private static final int VERSION = 1;
4     private static final String DATABASE_NAME = "crimeBase.db";
5
6     public CrimeBaseHelper(@Nullable Context context) {
7         super(context, DATABASE_NAME, null, VERSION);
8     }
9
10    @Override
11    public void onCreate(SQLiteDatabase sqLiteDatabase) {
12
13    }
14
15    @Override
16    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int
17                          i1) {
18    }
19 }

```

To access the database we can then call getWritableDatabase()

```

1 private CrimeLab(Context context) {
2     this.mContext = context.getApplicationContext();
3     this.mDatabase = new CrimeBaseHelper(mContext).
4         getWritableDatabase();
5 }

```

When we do this, SQLiteOpenHelper will:

- open up /data/data/com...../databases/thedatabasebeingopened.db
it will create a new database file if it does not already exist.
- If it is the first time the database has been created, call onCreate(...), then save out the latest version number.
- If it is not the first time, check the version number.
If the version number in CrimeBaseHelper is higher, call onUpgrade(...)

5.4 Writing to the Database

5.4.1 ContentValues

Writes and updates are done with ContentValues - a key-value store class, like Java's HashMap or Bundles.

Example of a helper function to create the instance of ContentValues for a row:

```

1 private static ContentValues getContentValues(@NonNull Crime crime)
2 {
3     ContentValues values = new ContentValues();
4     values.put(CrimeTable.Cols.UUID, crime.getId().toString());
5     values.put(CrimeTable.Cols.TITLE, crime.getTitle());
6     values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
7     values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);
8     return values;
9 }

```

5.4.2 Inserting rows

Can insert a new row to the database by using the content values object, and using the insert(...,...,...) method on the SQLite database object.

```

1 public void addCrime(Crime crime) {
2     ContentValues values = getContentValues(crime);
3     mDatabase.insert(CrimeTable.NAME, null, values);
4 }

```

5.4.3 Updating Rows

```

1 public void updateCrime(Crime crime) {
2     String uuidString = crime.getId().toString();
3     ContentValues values = getContentValues(crime);
4     mDatabase.update(CrimeTable.NAME, values,
5         CrimeTable.Cols.UUID + " = ?",
6         new String[] { uuidString });
7 }

```

To update a row, the same content values object is used from inserting; however, the update(...,...,...,...) method is called in the database object.

The third parameter is the where clause string which specifies what rows are updated. In this case, the UUID is used to identify the row.

To do this, the '?' syntax is used which tells the database to treat whatever string is in the following parameter as a pure string - not as SQL code. This prevents an SQL injection attack.

5.5 Reading from the Database

Reading from the database is done by using the query(...) function.

This returns a 'Cursor' object.

A cursor stores the retrieved data in key value pairs.

5.5.1 Retrieving a Cursor

```

1 public Cursor queryCrimes(String whereClause, String[] whereArgs) {
2     Cursor cursor = mDatabase.query(
3         CrimeTable.NAME,
4         null, // selects all columns
5         whereClause,

```

```

6     whereArgs,
7     null, // groupBy
8     null, // having
9     null // orderBy
10 );
11 return cursor;
12 }

```

5.5.2 Using a Cursor

To actually retrieve the returned data/values, the `get[Type]([Int])` function is used, where the `Int` is the key with the value of the column index, and the `Type` is the type of value which is stored.

To get the column index from the column name/title, the `getColumnIndex([String])` can be used.

```

1 String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
2 long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
3 int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));

```

It is cleaning, however, to use a custom wrapper of a cursor to encapsulate the cursor and retrieving of data withing one object.

Therefore create a class which extends `Cursor`

```

1 public class CrimeCursorWrapper extends CursorWrapper {
2
3     public CrimeCursorWrapper(Cursor cursor) {
4         super(cursor);
5     }
6
7     public Crime getCrime() {
8         String uuidString = this.getString(this.getColumnIndex(
9             CrimeTable.Cols.UUID));
10        String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
11
12        long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
13        int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));
14
15        Crime crime = new Crime(UUID.fromString(uuidString));
16        crime.setTitle(title);
17        crime.setDate(new Date(date));
18        crime.setSolved(isSolved != 0);
19
20        return crime;
21    }
22 }

```

From this point, convert the retrieved data into model objects.

To move the cursor along from one part of the query to the next, use the `Cursor.moveToFirst()` to move to the beginning of the query and `Cursor.moveToNext()` to move to the next position.

To check is the cursor is still inside the data set, using `Cursor.isAfterLast()` Hence the name, cursor.

```

1 public List<Crime> getCrimes() {
2     List<Crime> crimes = new ArrayList<>();

```



```

3  CrimeCursorWrapper cursorWrapper = queryCrimes(null, null);
4
5  try {
6      cursorWrapper.moveToFirst();
7      while (!cursorWrapper.isAfterLast()) {
8          crimes.add(cursorWrapper.getCrime());
9          cursorWrapper.moveToNext();
10     }
11 } finally {
12     cursorWrapper.close();
13 }
14
15 return crimes;
16 }

```

Remember to close the cursor.

If you don't the app will run out of open file handlers and the app will crash.

Example of retrieving specific row:

```

1  public Crime getCrime(UUID id) {
2      CrimeCursorWrapper cursor = queryCrimes(
3          CrimeTable.Cols.UUID + " = ?",
4          new String[] { id.toString() }
5      );
6
7      try {
8          if (cursor.getCount() == 0) {
9              return null;
10         }
11
12         cursor.moveToFirst();
13         return cursor.getCrime();
14     } finally {
15         cursor.close();
16     }
17 }

```

5.6 Deleting Rows

```

1  mDatabase.delete(CrimeTable.NAME,
2      CrimeTable.Cols.UUID + " = ?",
3      new String[] { crime.getId().toString() });

```

6 Implicit Intents

Implicit intents are used to start activities in another app.

In an implicit intent, you describe the job you require to be completed, and the OS will open an appropriate activity.

Compared to Explicit intents where you specify the class of the activity to start.

6.1 The Parts of an Implicit Intent

- action

Typically constants from the Intent class.

- location of any data
- type of data that the action is for
- optional categories

They can also include extras. However, there are not used by the OS to find the most appropriate

6.2 Advertising an Activity to Accept Implicit Intents

For example, to advertise an activity's capability to handle an implicit intent to open a web page, the following has to be declared within the AppManifest file.

```
1 <activity android:name=".BrowserActivity">
2   <intent-filter>
3     <action android:name="android.intent.action.VIEW"/>
4     <category android:name="android.intent.category.DEFAULT"/>
5     <data android:scheme="http" android:host="www.somewhere.com"/>
6   </intent-filter>
7 </activity>
```

6.3 Sending Text

6.3.1 With option of send

```
1 Intent intent = new Intent(Intent.ACTION_SEND);
2 intent.setType("text/plain");
3 intent.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
4 intent.putExtra(Intent.EXTRA_SUBJECT, getString(R.string.
5   crime_report_subject));
6 intent = Intent.createChooser(intent, getString(R.string.
7   send_report));
8 startActivity(intent);
```

6.4 Requesting Android for a Contact

6.4.1 The Request of Data

The implicit intent action will be Intent.ACTION_PICK.

Since we are expecting data to be sent back, the activity will be started with *startActivityForResult(...)* along with a request code to be able to identify the sent back response/result.

```
1 final Intent pickContactIntent = new Intent(Intent.ACTION_PICK,
2   ContactsContract.Contacts.CONTENT_URI);
3 startActivityForResult(pickContactIntent, REQUEST_CONTACT);
```

6.4.2 The Receiving of Data

Inside *onActivityResult(..., ..., ...)*

```
1  if (data == null) {
2      break;
3  }
4
5  Uri contactUri = data.getData();
6  String[] queryFields = new String[] {
7      ContactsContract.Contacts.DISPLAY_NAME
8  };
9  Cursor cursor = getActivity().getContentResolver().query(contactUri
10     , queryFields, null, null, null);
11  try {
12      if (cursor.getCount() == 0) {
13          return;
14      }
15      cursor.moveToFirst();
16      String suspect = cursor.getString(0);
17      mCrime.setSuspect(suspect);
18      mSuspectButton.setText(suspect);
19  } finally {
20      cursor.close();
21  }
```

6.4.3 Checking if the Device has a Contacts App

Use the OS Package manager to check if the device has a contacts app. If it does not, and you request data from the contacts app, the app will crash. It is therefore recommended to deactivate the functionality that uses requires this.

```
1  PackageManager packageManager = getActivity().getPackageManager();
2  if (packageManager.resolveActivity(pickContactIntent,
3      PackageManager.MATCH_DEFAULT_ONLY) == null) {
4      mSuspectButton.setEnabled(false);
5  }
```

This request returns an instance of *ResolveInfo* telling all about what the activity it found.

6.5 Taking Pictures with intents

All things Media related is defined in *MediaStore*. It defines the public interfaces used in Android for interacting with common media. The camera intent is defined in here as *MediaStore.ACTION_IMAGE_CAPTURE*.

By default *ACTION_IMAGE_CAPTURE* will take a thumb-nail picture and return it inside the *Intent* object returned in *onActivityResult(...)*. For a full-resolution picture, you need to tell it where to store the file on the file system.

This can be completed by passing a `Uri` pointing to where you want to save the file in `MediaStore.EXTRA_OUTPUT`. This `Uri` will point to a location serviced by `FileProvider`.

```
1 PackageManager packageManager = getActivity().getPackageManager();
2 mPhotoButton = (ImageButton) view.findViewById(R.id.crime_camera);
3 final Intent captureImageIntent = new Intent(MediaStore.
4     ACTION_IMAGE_CAPTURE);
5 boolean canTakePhoto = (mPhotoFile != null) && (captureImageIntent.
6     resolveActivity(packageManager) != null);
7 mPhotoButton.setEnabled(canTakePhoto);
8 mPhotoButton.setOnClickListener(new View.OnClickListener() {
9     @Override
10    public void onClick(View view) {
11        Uri uri = FileProvider.getUriForFile(getActivity(), "",
12            mPhotoFile);
13        captureImageIntent.putExtra(MediaStore.EXTRA_OUTPUT, uri);
14
15        List<ResolveInfo> cameraActivities = getActivity().
16            getPackageManager().queryIntentActivities(
17                captureImageIntent, PackageManager.MATCH_DEFAULT_ONLY);
18
19        for (ResolveInfo activity : cameraActivities) {
20            getActivity().grantUriPermission(activity.activityInfo.
21                packageName, uri, Intent.FLAG_GRANT_WRITE_URI_PERMISSION)
22                ;
23        }
24
25        startActivityResult(captureImageIntent, REQUEST_PHOTO);
26    }
27 });
```

6.5.1 Declaring the Camera Feature

To declare that the app uses the camera, add the following into `AndroidManifest.xml`:

```
1 <uses-feature android:name="android.hardware.camera"
2     android:required="false"/>
```

7 File Storage

Can store files in our own private storage for the app. This is the same location where the SQLite database is stored. So your app is the only one able to access them.

These files are accessed using the `Context` class.

The primary methods in the `Context` class:

- `getFilesDir()`

To let other apps access files stored in this location, we can use a `ContentProvider`. This allows us to expose content URIs to other apps; which in turn, allows us those apps to read and write to that specific URI.

7.1 FileProvider

When all that is required is to be able to receive a file from another application, a *FileProvider* can be used instead of implementing an entire *ContentProvider*; which in this case would be classed as overkill.

The first step is to declare a *FileProvider* as a *ContentProvider* hooked up to a specific authority. This is done by adding a content provider declaration to *AndroidManifest.xml*

The authority is a location where files can be saved to.

This gives other apps a target to write to.

```
1 <provider
2   android:authorities="com.bgsoftwarestudios.criminalintent.
   fileprovider"
3   android:name="android.support.v4.content.FileProvider"
4   android:exported="false"
5   android:grantUriPermissions="true" />
```

The *android : exported = "false"* attribute stops any app from using this provider except you and apps that you grant permission to.

And the *android : grantUriPermissions = "true"* attribute allows us to give other apps permission to write to this URI when we send out an intent.

7.1.1 Exposing/Telling the *FileProvider* what files it is exposing

To tell the *FileProvider* what files to expose, create a new resource file *xml/files.xml*.

You can create this initially by:

1. Right-click on the *app/res* directory.
2. Select new → Android resource file.
3. For Resource type select XML.
4. Enter *files* for the file name.
5. Enter the following into the the xml file

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <paths>
3   <files-path
4     name="crime_photos"
5     path="." />
6 </paths>
```

This declares the file paths that *FileProvider* will use/expose internally.

We then need to hook up the *files.xml* to the *FileProvider* by using a *meta-data* tag in *AndroidManifest.xml*.

7.1.2 Hooking up the paths description to the *FileProvider* within *AndroidManifest.xml*

Add a *meta-data* element inside the provider:

```
1 <provider
2   android:authorities="com.bgsoftwarestudios.criminalintent.
   fileprovider"
3   android:name="android.support.v4.content.FileProvider"
4   android:exported="false"
5   android:grantUriPermissions="true">
6   <meta-data
7     android:name="android.support.FILE_PROVIDER_PATHS"
8     android:resource="@xml/files"/>
9 </provider>
```

7.1.3 Revoking File write *FileProvider* permissions

When the picture is taken, the method *onActivityResult(..., ..., ...)* is called. So when the request comes back we can remove the file write permissions to that Uri for that external activity.

```
1 Uri uri = FileProvider.getUriForFile(getActivity(),
   FILE_PROVIDER_AUTHORITY, mPhotoFile);
2 getActivity().revokeUriPermission(uri, Intent.
   FLAG_GRANT_WRITE_URI_PERMISSION);
3 updatePhotoView();
```

8 Bitmaps

Bitmaps store image data as literal pixel data

8.1 Scaling Bitmaps

A 16-megapixel, 24-bit camera image compressed as a JPG with a size of 5 MB, would be 48 MB as a bitmap.

To shrink a bit map:

1. scan the file to determine the size
2. figure out how much it needs to be scaled for the desired dimensions.
3. reread the file and construct the new bitmap.

```
1 public class PictureUtils {
2
3   public static Bitmap getScaledBitmap(String path, int destWidth,
   int destHeight) {
4     // Read in the dimentions of the image on disk
5     BitmapFactory.Options options = new BitmapFactory.Options();
6     options.inJustDecodeBounds = true;
7     BitmapFactory.decodeFile(path, options);
```

```

8
9     float srcWidth = options.outWidth;
10    float srcHeight = options.outHeight;
11
12    // Calculate how much to scale down by
13    int inSampleSize = 1;
14    if (srcHeight > destHeight || srcWidth > destWidth) {
15        float heightScale = srcHeight / destHeight;
16        float widthScale = srcWidth / destWidth;
17        float scale = heightScale > widthScale ? heightScale :
18            widthScale;
19        inSampleSize = Math.round(scale);
20    }
21
22    options = new BitmapFactory.Options();
23    options.inSampleSize = inSampleSize;
24
25    // Read in and create final bitmap
26    return BitmapFactory.decodeFile(path, options);
27 }

```

The *inSampleSize* determines how big each new pixel is relative to each old pixel.

E.g. for a sample size of 2, one new pixel horizontally is equivalent to 2 old pixels horizontally; therefore, shrinking the bitmap horizontally by 2 (so the overall size will be one fourth).

8.1.1 Very conservative scaling

A very conservative scaling of bit map is to shrink it down to the size of an activity. This ensures that the image will never be too small for the size of the activity; therefore, keeping a high enough quality/resolution.

```

1 public static Bitmap getScaledBitmap(String path, @NonNull Activity
2     activity) {
3     Point size = new Point();
4     activity.getWindowManager().getDefaultDisplay().getSize(size);
5     return getScaledBitmap(path, size.x, size.y);
6 }

```

9 Strings

9.1 Plurals

```

1 <plurals name="subtitle_plural">
2   <item quantity="one">%1$d crime</item>
3   <item quantity="other">%1$d crimes</item>
4 </plurals>

```

To retrieve/use the string:

```

1 getResources().getQuantityString(R.plurals.subtitle_plural,
2     crimeSize, crimeSize);

```

9.2 Percentages

```
1 <string name="playback_speed_percentage">PlayBack Speed: %d%%</string>
```

9.3 Formatted Strings

9.3.1 Strings with values embedded

```
1 <string name="welcome_messages">Hello, %1$s! You have %2$d new messages.</string>
```

```
1 String text = getString(R.string.welcome_messages, username, mailCount);
```

Use `%1$s` for *strings* and `%1$d` for decimals.

10 Intents

10.1 Starting an activity in a new task

```
1 ActivityInfo activityInfo = resolveInfo.activityInfo;
2 Intent intent = new Intent(Intent.ACTION_MAIN)
3     .setClassName(activityInfo.applicationInfo.packageName, activityInfo.name)
4     .addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
5 startActivity(intent);
```

11 Two-Pane Master-Detail

This is most commonly used for tablets.

However, you will commonly require different layout for on phone vs tablet.

To do this, use an alias resource.

11.1 Alias Resource

In *res/values*, create a resource file for the default (phone) and list the layout to be used. Call this file *refs.xml*.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <item name="activity_masterdetail" type="layout">@layout/activity_fragment</item>
4 </resources>
```

Then for the layouts that are needed to be used for larger screen sizes include the Smallest Screen Width qualifier with a value of 600.

Call this file *refs.xml* also.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <item name="activity_masterdetail" type="layout">@layout/activity_twopane</item>
4 </resources>
```


This will display the later when the minimum screen dimension is 600dp.
To use this in code, refer to the layout you want using the name of the item.
The file/layout that is then displayed/used is declared as the string in the *item* element.

11.2 Determining Device size

page 335.

12 RecyclerView

12.1 Item cell padding

```
1 package com.bengavan.android.recyclerviewtesting;
2
3 import android.graphics.Rect;
4 import android.view.View;
5
6 import androidx.annotation.NonNull;
7 import androidx.recyclerview.widget.RecyclerView;
8
9 public class RecyclerViewItemSpacing extends RecyclerView.
    ItemDecoration {
10
11     private int left = 0;
12     private int top = 0;
13     private int right = 0;
14     private int bottom = 0;
15
16     public RecyclerViewItemSpacing(int spacing) {
17         new RecyclerViewItemSpacing(spacing, spacing, spacing, spacing)
18         ;
19     }
20
21     public RecyclerViewItemSpacing(int left, int top, int right, int
        bottom) {
22         this.left = left;
23         this.top = top;
24         this.right = right;
25         this.bottom = bottom;
26     }
27
28     @Override
29     public void getItemOffsets(@NonNull Rect outRect, @NonNull View
        view, @NonNull RecyclerView parent, @NonNull RecyclerView.
        State state) {
30         outRect.left = left;
31         outRect.top = top;
32         outRect.right = right;
33         outRect.bottom = bottom;
34     }
35 }
```

12.2 Swipe to delete

The main components to fulfill this:

- *ItemTouchHelper*

This is attached to the *RecyclerView* and watches for swipes

- *ItemTouchHelper.SimpleCallback*

This is passed into *ItemTouchHelper*'s constructor and contains the code that will run once the item is swiped.

12.2.1 Creating *ItemTouchHelper.SimpleCallback*

Create a new class that extends *ItemTouchHelper.SimpleCallback*:

```
1 public class SwipeToDeleteCallback extends ItemTouchHelper.  
    SimpleCallback {  
2  
3     private Adapter adapter;  
4  
5     public SwipeToDeleteCallback(Adapter adapter) {  
6         super(0, ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT);  
7         this.adapter = adapter;  
8     }  
9 }
```

The first parameter in the *super(...)* call adds support for dragging the item up or down; so leaving it *0* will prevent this.

The send parameter is for left and right, which we do want both; hence the or of both left and right.

We also take in a reference to the adapter for the *RecyclerView*.

We then need to override two methods:

```
1 @Override  
2 public boolean onMove(@NonNull RecyclerView recyclerView, @NonNull  
    RecyclerView.ViewHolder viewHolder, @NonNull RecyclerView.  
    ViewHolder target) {  
3     return false;  
4 }  
5  
6 @Override  
7 public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder,  
    int direction) {  
8  
9 }
```

The *onMove(...)* override is not required; however, it is required to be overridden.

The *onSwiped(...)* override is what we are going to use to implement swipe to delete.

Creating a background for when the user swipes:

```

1  @Override
2  public void onChildDraw(@NonNull Canvas c, @NonNull RecyclerView
    recyclerView, @NonNull RecyclerView.ViewHolder viewHolder,
    float dX, float dY, int actionState, boolean isCurrentlyActive)
    {
3      super.onChildDraw(c, recyclerView, viewHolder, dX, dY,
        actionState, isCurrentlyActive);
4
5      View itemView = viewHolder.itemView;
6
7      int backgroundCornerOffset = 20; // So the background is behind
        the rounded corners of itemView
8
9      int iconMargin = (itemView.getHeight() - icon.getIntrinsicHeight
        ()) / 2;
10     int iconTop = itemView.getTop() + iconMargin;
11     int iconBottom = iconTop + icon.getIntrinsicHeight();
12
13     if (dX > 0) { // Swiping to the right
14         int iconLeft = itemView.getLeft() + iconMargin + icon.
            getIntrinsicWidth();
15         int iconRight = itemView.getLeft() + iconMargin;
16
17         icon.setBounds(iconLeft, iconTop, iconRight, iconBottom);
18
19         background.setBounds(itemView.getLeft(), itemView.getTop(),
20             itemView.getLeft() + ((int) dX) + backgroundCornerOffset,
21             itemView.getBottom());
22
23     } else if (dX < 0) { // Swiping to the left
24         int iconLeft = itemView.getRight() - iconMargin - icon.
            getIntrinsicWidth();
25         int iconRight = itemView.getRight() - iconMargin;
26         icon.setBounds(iconLeft, iconTop, iconRight, iconBottom);
27         background.setBounds(itemView.getRight() + ((int) dX) + -
            backgroundCornerOffset,
28             itemView.getTop(), itemView.getRight(), itemView.getBottom());
29     } else { // View is un-swiped
30         background.setBounds(0,0,0,0);
31     }
32
33     background.draw(c);
34     icon.draw(c);
35 }

```

12.3 Press and hold to change order

12.4 Rounded Corners

Add a drawable resource (in *res/drawable/*):

```

1  <shape xmlns:android="http://schemas.android.com/apk/res/android"
2      android:shape="rectangle">
3
4      <corners android:radius="5dp"/>
5

```

```

6   <solid android:color="@android:color/holo_blue_light" />
7
8 </shape>

```

then set it as the background of the list item layout.

13 TextView

14 Localization

See chapter 18

15 Accessibility

15.1 TalkBack

TalkBack is an Android screen reader made by Google.

See Chapter 19.

16 Styles and Themes

16.1 Styles

Declared inside the *styles.xml* file

```

1 <style name="BeatBoxButton">
2   <item name="android:background">@color/dark_blue</item>
3 </style>

```

16.2 Style Inheritance

The second style declaration of *BeatBoxButton.Strong* will inherit all of the items declared in the style declared above it - *BeatBoxButton*.

```

1 <style name="BeatBoxButton">
2   <item name="android:background">@color/dark_blue</item>
3 </style>
4
5 <style name="BeatBoxButton.Strong">
6   <item name="android:textStyle">bold</item>
7 </style>

```

16.3 Themes

Are applied to all objects across the app.

16.3.1 android:windowBackground

Changes the color of the background.

Need to use the android name space since the 'windowBackground' attribute that we are overriding, is declared in the Android OS.

16.3.2 Overriding themes

Specifying the parent in the style name only works when the parent theme exists in the same package.

So specify the parent in the name when it exists in the same package (one of your own). But when it crosses over to a different package (e.g. Android OS, AppCompatActivity), use the explicit *parent = ""* attribute.

16.3.3 Accessing Theme Attributes

To access attributes declared in a theme, e.g. to access a color declared in a them, use the ? syntax:

```
1 android:background="?attr/colorAccent "
```

This retrieve the color that the attribute 'colorAccent' in the theme points to.

17 Assets

17.1 Why Assets over Dependencies

More basic/less overhead.

For sound files, we can store them in the 'res/raw' folder withing

The resource system is limited to a flat hierarchy, unlike assets which can implement its own custom file structure. This therefore is more organized when there are large amounts of assets.

Resources do not allow you to read in multiple files at once - you have to refer to each file independently (unlike assets where we can get the file list for a folder and then loop through each asset, retrieving each one at a time). Resources are given ids such as R.raw.file.

17.2 Creating an Assets Folder

To create an assets folder for your app:

- Go to the Android option for the folder/files layout
- Right-click → New → Folder (Last section of the menu with the android guy as the symbol to the left of the options)→ Assets Folder.
- Keep 'Change Folder Location' unchecked.

- select 'main' for 'Target Source Set'
- Press finish.
- Proceed to create sub directories for organization of your assets.

17.3 Accessing Assets

Assets are accessed using the *AssetsManager* class.

17.3.1 Getting an *AssetsManager*

You get get at *AssetsManager* from any context.

17.3.2 Getting assets files names

```

1 private void loadSounds() {
2     String[] soundNames;
3     try {
4         soundNames = assetManager.list(SOUNDS_FOLDER);
5         Log.i(TAG, "Found " + soundNames.length + " sounds");
6     } catch (IOException ioe) {
7         Log.e(TAG, "Could not list assets", ioe);
8         return;
9     }
10 }
```

where *SOUNDS_FOLDER* the directory within the Assets folder that you want to access.

17.4 SoundPool

SoundPool can load lots of sound files into memory and control the maximum number of sounds that are playing back at one time.

A benefit of *SoundPool* over other methods of playing sounds is that you ask it to play, there is very little lag and starts to play almost immediately.

A trade off is that you are required to load the sound before it is played.

17.4.1 Creating a SoundPool

The *SoundPool* constructor takes the maximum number of sounds that can be played at any one time, the type of *AudioManager* you require (for music use *STREAM_MUSIC*), and the third is the sample rate converter (the documentation says it is ignored).

```

1 soundPool = new SoundPool(MAX_SOUNDS, AudioManager.STREAM_MUSIC, 0)
    ;
```

If you are playing the maximum number of sounds and then you try to play another, the oldest sound will be stopped to make space for the new sound being added.

The *AudioManager*.*** also specifies what audio volume is adjusted when it is playing (specifies what volume is used to play that sound).

17.4.2 Loading Sounds

Give each sound a unique ID - typically an integer value. Use *Integer* so that it can have an unspecified value of *null*.

```
1 private void load(@NotNull Sound sound) throws IOException {
2     AssetFileDescriptor assetFileDescriptor = assetManager.openFd(
3         sound.getAssetPath());
4     int soundId = soundPool.load(assetFileDescriptor, 1); // Loads a
5     sound.setSoundId(soundId);
6 }
```

soundPool.load(...) loads a file into *soundPool* for later playback. It also returns an ID to keep track of it so it is able to play it or unload it at a later time.

assetManager.openFd(...) throws the *IOException*.

17.4.3 Playing Sounds

```
1 soundPool.play(soundId, 1f, 1f, 1, 0, 1f);
```

'priority' is ignored.

For 'loop', '-1' causes it to loop forever. 0 = do not loop.

18 XML Drawables

- not density specific, so are placed/stored in the default drawable folder instead of a density-specific one.

18.1 What is a Drawable

Android calls anything that is intended to be drawn on the screen a drawable.

18.2 State List Drawables

Is a drawable resource file that points to other drawable resource files for different states. This file is set as the resource on the object.

For example, a button with two states - pressed and normal (not pressed):

In the resource file that is allocated to the object is a selector with 2 items - one for each of the states:

```
1 <selector xmlns:android="http://schemas.android.com/apk/res/android"
2     ">
3     <item android:drawable="@drawable/button_beat_box_pressed"
4         android:state_pressed="true"/>
5     <item android:drawable="@drawable/button_beat_box_normal"/>
6
7 </selector>
```

Then in the first file that is pointed to:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <shape xmlns:android="http://schemas.android.com/apk/res/android"
3     android:shape="oval">
4
5     <solid
6         android:color="@color/red"/>
7
8 </shape>

```

and in the second:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <shape xmlns:android="http://schemas.android.com/apk/res/android"
3     android:shape="oval">
4
5     <solid
6         android:color="@color/dark_blue"/>
7
8 </shape>

```

For the button, there are different states, including disabled, focused, and activated.

18.3 shape drawables

18.4 layer list drawables

Allows two XML drawables to be combined into one.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <layer-list xmlns:android="http://schemas.android.com/apk/res/
3     android">
4
5     <item>
6         <shape
7             android:shape="oval">
8
9             <solid
10                 android:color="@color/red"/>
11
12             </shape>
13         </item>
14
15         <item>
16             <shape
17                 android:shape="oval">
18
19                 <stroke
20                     android:width="4dp"
21                     android:color="@color/dark_red"/>
22             </shape>
23         </item>
24 </layer-list>

```


19 Custom Views

Two broad categories:

- Simple
 - Has no child sub-views (even though it may be complex in of it self).
 - Will almost always perform custom rendering.
 - Subclass *View*
- Composite
 - Composed by other view objects.
 - Typically manage child views but do not perform custom rendering.
 - Rendering is delegated to each child view.
 - Subclass an appropriate layout class, such as *FrameLayout*

19.1 Custom Simple View

A basic skeletal custom simple view:

```
1 import android.content.Context;
2 import android.util.AttributeSet;
3 import android.view.View;
4
5 public class BoxDrawingView extends View {
6
7     // Used when inflating view in code
8     public BoxDrawingView(Context context) {
9         super(context);
10    }
11
12    // Used when inflating view from xml
13    public BoxDrawingView(Context context, AttributeSet attrs) {
14        super(context, attrs);
15    }
16 }
```

19.1.1 Background Color

To get the color for *resources*

```
1 ResourcesCompat.getColor(getResources(), R.color.your_color,
    your_theme);
```

where *your_theme* can be *null*.

19.2 Handling Touch Events

One way to implement touch listening is to implement the *View.OnTouchListener* interface by using the method:

```
1 public void setOnTouchListener(OnTouchListener l)
```

However, since the view is a subclass of *View*, we can override the *View* method:

```
1 public boolean onTouchEvent(MotionEvent event)
```

19.2.1 *MotionEvent*

The *MotionEvent* includes its location and its action. The action describes the stage of the event and can have the following values:

Action Constants	Description
ACTION_DOWN	User's finger touches the screen
ACTION_MOVE	User' moves finger on the screen
ACTION_UP	User lifts finger off the screen
ACTION_CANCEL	A parent view has intercepted the touch event

This can be retrieved from the event by calling:

```
1 public final int getAction()
```

Basic touch event handling:

```
1 @Override
2 public boolean onTouchEvent(MotionEvent event) {
3     PointF current = new PointF(event.getX(), event.getY());
4     String action = "";
5
6     switch (event.getAction()) {
7         case MotionEvent.ACTION_DOWN:
8             action = "ACTION_DOWN";
9             break;
10        case MotionEvent.ACTION_MOVE:
11            action = "ACTION_MOVE";
12            break;
13        case MotionEvent.ACTION_UP:
14            action = "ACTION_UP";
15            break;
16        case MotionEvent.ACTION_CANCEL:
17            action = "ACTION_CANCEL";
18            break;
19    }
20
21    Log.i(TAG, "onTouchEvent: " + action + " at: (" + current.x + ",
22            " + current.y + ")");
23    return true;
24 }
```

19.3 Rendering inside *onDraw(Canvas)*

The two main classes are:

- *Canvas*

Has all the drawing operations you can perform.
line, circle, word, or a rectangle

- *Paint*

Determines how these operations are performed.

Whether shapes are filled, what fonts are used, and what.

20 Animation

There are three properties:

- View rotation
- View scaling
- View translation

20.1 Interpolation

20.1.1 Setting the interpolator

```
1 heightAnimator.setInterpolator(new AccelerateInterpolator());
```

21 9-Patch Images

See Page 450

22 AsyncTasks

22.1 *AsyncTasks.cancel(boolean)*

22.1.1 *.cancel(true)*

.cancel(true) is the more severe way of stopping the task.

Will just interrupt the thread.

not advisable, should avoid it if you can

22.1.2 *.cancel(false)*

The better way of doing things.

Just sets *isCancelled()* to true. So the *AsyncTask* can then check *isCancelled()* inside *doInBackground(...)* and elect to finish prematurely.

22.2 message queue

A *message loop* consists of a *thread* and a *looper*

22.3 Looper

The *looper* is the object that manages a thread's message queue.

Multiple *handlers* can be attached to one *looper*.

22.4 Handler

22.5 Background Thread

22.5.1 Assembling a Background Thread

22.6 *ConcurrentHashMap*

ConcurrentHashMap is a thread-safe version of *HashMap*.

23 HTTP

!!!!!!!!!!!!!!!!!!!!!!YOU NEED TO HAVE THE INTERNET PERMISSION!!!!!!!!!!!!!!!!!!!!!!

23.1 Make get requests

```
1 public byte[] getUrlBytes(String urlString) throws IOException {
2     URL url = new URL(urlString);
3     HttpURLConnection connection = (HttpURLConnection) url.
        openConnection();
4
5     try {
6         ByteArrayOutputStream outputStream = new ByteArrayOutputStream
            ();
7         InputStream inputStream = connection.getInputStream();
8
9         if (connection.getResponseCode() != HttpURLConnection.HTTP_OK)
10            {
11                throw new IOException(connection.getResponseMessage() + ":
                    with " + urlString);
12            }
13
14         int bytesRead = 0;
15         byte[] buffer = new byte[1024];
16         while ((bytesRead = inputStream.read(buffer)) > 0) {
17             outputStream.write(buffer, 0, bytesRead);
18         }
19         outputStream.close();
20         return outputStream.toByteArray();
21     } finally {
22         connection.disconnect();
23     }
24 }
25
26 public String getUrlString(String urlString) throws IOException {
27     return new String(getUrlBytes(urlString));
28 }
```

23.2 Connecting to local sever from emulator

Use the IP address of *10.0.2.2* instead of *localhost*

This is due to that when using *localhost*, it is trying to connect to a that address on the actual emulator.

23.3 building a Url String

```
1 try {
2     String url = Uri.parse("https://www.flickr.com/services/rest/")
3         .buildUpon()
4         .appendQueryParameter("method", "flickr.photos.getRecent")
5         .appendQueryParameter("api_key", API_KEY)
6         .appendQueryParameter("format", "json")
7         .appendQueryParameter("nojsoncallback", "1")
8         .appendQueryParameter("extras", "url_s")
9         .build().toString();
10    String jsonString = getUrlString(url);
11    Log.i(TAG, "fetchItems: Retrieved JSON: " + jsonString);
12 } catch (IOException ioe) {
13     Log.e(TAG, "fetchItems: Failed to fetch items", ioe);
14 }
```

23.4 Parsing JSON

24 *LruCache*

A cache of a defined size of the format of a key-value map.

24.1 Creating a new Cache

```
1 private final LruCache<K, V> thumbnailCache;

of size 40MiB:

1 int cacheSize = 40 * 1024 * 1024; // 40MiB
2 thumbnailCache = new LruCache<String, Bitmap>(cacheSize) {
3     @Override
4     protected int sizeOf(String key, Bitmap value) {
5         return value.getByteCount();
6     }
7 };
```

24.2 Retrieval from the Cache

```
1 V value = cache.get(key)
```

The object retrieved is already of the type that was *put* and therefore does not require casting.

24.3 Adding item to cache

```
1 private void addToCache(String key, Bitmap bitmap) {
2     synchronized (thumbnailCache) {
3         if (thumbnailCache.get(key) == null) {
4             thumbnailCache.put(key, bitmap);
5         }
6     }
7 }
```

To be thread-safe, the cache object has to be declared as final with any putting/getting carried out within the scope of *synchronized*.

Before adding, it will be advisable for most implementations to check that there is not a value already stored for that key; otherwise, it will be overridden.

25 *SearchView*

A *SeachView* is an *action view* class.

It can be embedded into any *view* including a *ToolBar*.

26 *SharedPreferences*

Should use

```
1 PreferenceManager.getDefaultSharedPreferences(context)
```

from the *androidx* library instead of

```
1 content...
```

This will use the default settings which include that the preferences are only accessible from within the app.

```
1 public class QueryPreferences {
2
3     private static final String TAG = "QueryPreferences";
4     private static final String PREF_SEARCH_QUERY = "searchQuery";
5
6     public static String getStoredQuery(Context context) {
7         return PreferenceManager.getDefaultSharedPreferences(context)
8             .getString(PREF_SEARCH_QUERY, null);
9     }
10
11     public static void setStoredQuery(Context context, String query)
12     {
13         PreferenceManager.getDefaultSharedPreferences(context)
14             .edit()
15             .putString(PREF_SEARCH_QUERY, query)
16             .apply();
17     }
18 }
```

27 *Background Services*

To be able to run code/do something in the background not requiring a view, a *service* is required.

Service is a subclass of *Context*

27.1 *IntentService*

An *IntentService* is probably the most common service.

27.1.1 Basic Skeletal example

```
1 public class PollService extends IntentService {
2     private static final String TAG = "PollService";
3
4     public static Intent newIntent(@NonNull Context context) {
5         return new Intent(context, PollService.class);
6     }
7
8     public PollService() {
9         super(TAG);
10    }
11
12    @Override
13    protected void onHandleIntent(Intent intent) {
14        Log.i(TAG, "onHandleIntent: Recieved an Intent: " + intent);
15    }
16 }
```

27.2 *commands*

A service's intents are called *commands*.

Each *command* is an instruction for the service to do something.

Since services respond to intents, they must be declared in *AndroidManifest.xml*, just like activities do. Like activities

27.3 Safe background networking

Since you are carrying out networking in the background, android provides an option to disable background networking to save on power consumption and performance improvements.

As a consequence of this, you should always check with *ConnectivityManager* that the network is available.

27.3.1 Checking for background networking availability

```
1 //need to figure out undeprecated method
2 private boolean isNetworkAvailableandConnected() {
3     ConnectivityManager cm = (ConnectivityManager) getSystemService(
4         CONNECTIVITY_SERVICE);
5     // TODO: Use un-deprecated method then add to LATEX document
6     boolean isNetworkAvailable = (cm.getActiveNetworkInfo() != null);
7     boolean isNetworkConnected = (isNetworkAvailable && cm.
8         getActiveNetworkInfo().isConnected());
9     return isNetworkConnected;
10 }
```

27.4 *AlarmManager*

AlarmManager is a system service that can send intents for you.

If you used *handlers* instead, if the user closes the app (i.e. navigates away from all activities of the app) the system will shutdown that process which also

results in all of the *Handler's messages* to go kaput with it.

To request *AlarmPanager* to send an intent, a *PendingIntent* is used which is sent to other components of the system, such as *AlarmManager*.

You will want to turn the alarm on and off from the UI via an activity or a fragment.

28 Notifications

28.1 Creating a notification

```
1 Resources resources = getResources();
2 Intent photoGalleryIntent = PhotoGalleryActivity.newIntent(this);
3 PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
    photoGalleryIntent, 0);
4
5 Notification notification = new NotificationCompat.Builder(this)
6     .setTicker(resources.getString(R.string.new_pictures_title))
7     .setSmallIcon(android.R.drawable.ic_menu_report_image)
8     .setContentTitle(resources.getString(R.string.
    new_pictures_title))
9     .setContentText(resources.getString(R.string.new_pictures_text)
    )
10    .setContentIntent(pendingIntent)
11    .setAutoCancel(true)
12    .build();
13
14 NotificationManagerCompat notificationManager =
    NotificationManagerCompat.from(this);
15 notificationManager.notify(0, notification);
```

29 Broadcast Intents

29.1 Regular *Intents* vs *Broadcast Intents*

Broadcast intents are like regular intents; however, unlike regular intents, *Broadcast Intents* can be sent once and received by multiple components, called *broadcast receivers*.

Even though *broadcast receivers* can receive explicit intents, they are very rarely used in this manner. Instead, they are used to respond to *broadcast intents*. This is put down to that *explicit intents* can only have one receiver.

29.2 Receiving a system Broadcast

29.2.1 Waking Up on Boot

You can detect when a boot (turned on) is completed by listening for a broadcast intent with the *BOOT_COMPLETED* action.

This can be listened for by creating and registering a standalone *broadcast receiver* that filters for the appropriate action.

29.2.2 Creating and Registering a standalone receiver

A *standalone receiver* is declared in the *AndroidManifest.xml*. This means that it can be activated, even if the app process is dead. This is compared to a *dynamic receiver* where it is tied to the life-cycle of the app.

If a *standalone receiver* is not registered with the system (via *AndroidManifest.xml*), just like an activity or service, it will not be able to receive any intents.

Basic Example of a *standalone receiver*:

```
1 content...
```

Registration of a *standalone receiver* inside the *application* tag:

```
1 <receiver android:name=".StartupReceiver">
2   <intent-filter>
3     <action android:name="android.intent.action.BOOT_COMPLETED" />
4   </intent-filter>
5 </receiver>
```

Also requires the permission:

```
1 <uses-permission android:name="android.permission.
  RECEIVE_BOOT_COMPLETED" />
```

29.3 Sending Broadcast Intents

```
1 private static final String ACTION_SHOW_NOTIFICATION = "com.
  bengavan.photogallery.SHOW_NOTIFICATION";
```

```
1 sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION));
```

29.4 *Dynamic Broadcast Receiver*

When you create a *dynamic receiver*, it should be registered on the fragment start and unregistered when the fragment is ended

29.4.1 Register

Dynamic Broadcast Receivers are registered by calling *registerReceiver(BroadcastReceiver)*.

```
1 public abstract class VisibleFragment extends Fragment {
2
3   private static final String TAG = "VisibleFragment";
4
5   @Override
6   public void onStart() {
7     super.onStart();
8     IntentFilter intentFilter = new IntentFilter(PollService.
9       ACTION_SHOW_NOTIFICATION);
10    getActivity().registerReceiver(onShowNotification, intentFilter
11    );
12  }
13
14   @Override
```

```

13 public void onStop() {
14     super.onStop();
15     getActivity().unregisterReceiver(onShowNotification);
16 }
17
18 private BroadcastReceiver onShowNotification = new
19     BroadcastReceiver() {
20         @Override
21         public void onReceive(Context context, Intent intent) {
22             Toast.makeText(getActivity(), "Got a broadcast: " + intent.
23                 getAction(), Toast.LENGTH_LONG).show();
24         }
25     };
26 }

```

So that only your app can receive these broadcasts, create a permission and pass it to the broadcaster and receiver.

To define a permission in *AndroidManifest.xml*

```

1 <permission android:name="com.bengavan.photogallery.PRIVATE"
2     android:protectionLevel="signature" />

1 getActivity().registerReceiver(onShowNotification, intentFilter,
    PollService.PERM_PRIVATE, null);

```

29.4.2 Un-Register

Dynamic Broadcast Receiver is unregistered by calling *unregisterReceiver(BroadcastReceiver)*.

29.5 IntentFilter

IntentFilter can be created in code as well as expressed in xml by using *addCategory(String)*, *addAction(String)*, *addDataPath(String)*, and so on.

```

1 IntentFilter intentFilter = new IntentFilter(PollService.
    ACTION_SHOW_NOTIFICATION);

```

29.6 For longer running tasks

For longer tasks, you can still use a broadcast receiver; however, in that small window of opportunity, start a service (which can run as long as required to complete the task) (it can also go off the main thread)

Can use *BroadcastReceiver.goAsync()* to ...

30 Location

30.1 Requirements

To use Google play services, you need to have the following dependency:

```

1      implementation 'com.google.android.gms:play-services-location
      :17.0.0'

```

Google play services are available/stored in the Google Play store app. Therefore, Google Play Services can only work with apps that are installed on devices with the play store app installed. Since most apps are installed through the play store, it is almost a guarantee that the app will be installed on those devices. It is however, advisable to check that Play services are available.

30.1.1 Checking Play Services are Available

```

1  private static final int REQUEST_ERROR = 0;
2
3  @Override
4  protected void onResume() {
5      super.onResume();
6
7      GoogleApiAvailability apiAvailability = GoogleApiAvailability.
        getInstance();
8      int errorCode = apiAvailability.isGooglePlayServicesAvailable(
        this);
9
10     if (errorCode != ConnectionResult.SUCCESS) {
11         Dialog errorDialog = apiAvailability.getErrorDialog(this,
            errorCode, REQUEST_ERROR, new DialogInterface.
            OnCancelListener() {
12             @Override
13             public void onCancel(DialogInterface dialogInterface) {
14                 finish();
15             }
16         });
17         errorDialog.show();
18     }
19 }

```

30.1.2 Location Permissions

For location requests, there are two types: *FINE_LOCATION* and *COARSE_LOCATION*. Fine location uses GPS and Coarse location uses Wi-Fi and cell towers.

To access these, the following permissions are required:

```

1  <uses-permission android:name="android.permission.
    ACCESS_FINE_LOCATION" />
2  <uses-permission android:name="android.permission.
    ACCESS_COARSE_LOCATION" />
3  <uses-permission android:name="android.permission.INTERNET" />

```

Both *FINE_LOCATION* and *COARSE_LOCATION* are dangerous permissions; therefore, a run-time request is required.

30.1.3 GoogleApiClient

To get a client withing *onCreate(Bundle)*:

```

1 client = new GoogleApiClient.Builder(getActivity())
2     .addApi(LocationServices.API)
3     .build();

```

You should connect and disconnect in *onStart()* and *onStop()* respectively.

```

1 @Override
2 public void onStart() {
3     super.onStart();
4     client.connect();
5 }
6
7 @Override
8 public void onStop() {
9     super.onStop();
10    client.disconnect();
11 }

```

If the app is not connected to a client, it will not be able to do anything location related.

If you want to get callbacks when the connection to the client is successful, suspended, or failed; add a *ConnectionCallbacks*:

```

1 client = new GoogleApiClient.Builder(getActivity())
2     .addApi(LocationServices.API)
3     .addConnectionCallbacks(new GoogleApiClient.ConnectionCallbacks
4         () {
5         @Override
6         public void onConnected(@Nullable Bundle bundle) {
7             getActivity().invalidateOptionsMenu();
8         }
9
10        @Override
11        public void onConnectionSuspended(int i) {
12        }
13    })
14    .addOnConnectionFailedListener(new GoogleApiClient.
15        OnConnectionFailedListener() {
16        @Override
17        public void onConnectionFailed(@NonNull ConnectionResult
18            connectionResult) {
19            Log.i(TAG, "onConnectionFailed: GoogleApiClient connection
20                failed: " + connectionResult.toString());
21        }
22    })
23    .build();

```

30.2 Getting a Location Fix

To get the location, a *LocationRequest* is used. It has the following parameters:

Parameters	Description
<i>interval</i>	how frequently the location should be updated
<i>number of updates</i>	how many times the location should be updated
<i>priority</i>	How Android should priorities battery life against accuracy to satisfy your request
<i>expiration</i>	whether the request should expire, and if so, when
<i>smallest displacement</i>	The smallest amount the devices should move (in meters) to trigger a location update

```
1 content...
```

30.2.1 Checking You have location permissions

Since *FINE_LOCATION* and *COARSE_LOCATION* are both grouped under the location permission, you only have to check that you have permission for one.

```
1 private boolean hasLocationPermission() {
2     int result = ContextCompat.checkSelfPermission(getActivity(),
3         LOCATION_PERMISSIONS[0]);
4     return (result == PackageManager.PERMISSION_GRANTED);
}
```

30.2.2 Requesting location permissions

To request for location permissions at run-time, call the method:

```
1 public final void requestPermissions(@NonNull String[] permissions,
2     int requestCode)
```

where the string array is an array of permission strings. All permissions that can be declared in *AndroidManifest.xml* can be accessed through *Manifest.permission.****.

For example:

```
1 private static final String[] LOCATION_PERMISSIONS = new String[] {
2     Manifest.permission.ACCESS_FINE_LOCATION,
3     Manifest.permission.ACCESS_COARSE_LOCATION
4 };
```

This method call has a callback to the method:

```
1 public void onRequestPermissionsResult(int requestCode, @NonNull
2     String[] permissions,
3     @NonNull int[] grantResults)
```

which can be overridden

```
1 @Override
2 public void onRequestPermissionsResult(int requestCode, @NonNull
3     String[] permissions, @NonNull int[] grantResults) {
4     switch (requestCode) {
5         case REQUEST_LOCATION_PERMISSIONS:
6             if (hasLocationPermission()) {
```

```

6         findImage();
7     }
8     default:
9         super.onRequestPermissionsResult(requestCode, permissions,
10             grantResults);
11 }

```

You should check the request code, and if it matches, check that you have location permission (shorter than checking the response).

30.3 Requesting Location

```

1 private void findImage() throws SecurityException {
2     LocationRequest request = LocationRequest.create();
3     request.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
4     request.setNumUpdates(1);
5     request.setInterval(0);
6     LocationServices.FusedLocationApi.requestLocationUpdates(client,
7         request, new LocationListener() {
8         @Override
9         public void onLocationChanged(Location location) {
10             Log.i(TAG, "onLocationChanged: Got a fix: " + location);
11         }
12     });
13 }

```

// NOTE: *LocationServices.FusedLocationApi* is deprecated.

31 Material Design

32 StrictMode

To activate *StrictMode*'s recommended policies, call:

```
1 StrictMode.enableDefaults();
```

These include:

- networking on the main thread.
- disk reads and writes on the main thread
- activities kept alive beyond their natural life-cycle
Also known as an *activity leak*
- un-closed SQLite database cursors
- clear-text network traffic not wrapped in SSL/TLS

33 Unit Testing

33.1 Creating a Test Class

1. Go to the class you want to test.
2. 'command + shift + T' to navigate to the test class
3. if there is no test class, select 'Create New Test...'
4. Select 'JUnit4'
5. set 'setUp/@Before' checked
6. Keep everything else unchecked.
7. Press OK.
8. Choose between 'androidTest' and 'test'

33.1.1 *androidTest* vs *test*

androidTests are run at run-time on a device or emulator withing the android environment.

tests are run the the development machine outside of the android environment.

tests tend to be faster

33.2 Setting Up the Test

33.2.1 Using Mocked Dependencies

33.3 Writing Tests

34 Refactoring Techniques and Tools

There are many techniques and tools that can be used to make refactoring code easier.

34.1 Extracting a method with Android Studio

1. Highlight the code that you want to be extracted
2. Right-click and select Refactor → Extract → Method
3. Set the Visibility and method Name
4. Press Refactor or Preview to preview the changes

5. If there are multiple occurrences of the highlighted text being extracted, android studio will ask if you want to replace these as well

You can either replace each occurrence one by one, or by choosing all.

35 Building the APK

35.1 Separate APK for Screen size

Since all the files in the Drawable directory is built into the final APK, it can provide ALOT of bloat to the final app size.

Therefore, we can generate separate APKs for each screen density, containing only the necessary resources.

See `configure-apk-splits`

35.1.1 Exception

The only exception to this, every density of the launcher icon is maintained.