Android

Ben Gavan

August 26, 2019

Contents

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();
```

```
                Date date = new GregorianCalendar(year, month, day)
                    .getTime();
                sendResult(Activity.RESULT_OK, date);
              }
            })
            .create();
  }

  private void sendResult(int resultCode, Date date) {
      if (getTargetFragment() == null) {
        return;
      }

      Intent intent = new Intent();
      intent.putExtra(EXTRA_DATE, date);

      this.getTargetFragment().onActivityResult(this.
          getTargetRequestCode(), resultCode, intent);
  }
}
```

2.2 Setting a Target Fragment

When displaying a dialog view from a fragment, we need to create a relationship between them se 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:

```
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.

```
private void sendResult(int resultCode, Date date) {
  if (getTargetFragment() == null) {
    return;
  }

  Intent intent = new Intent();
  intent.putExtra(EXTRA_DATE, date);

```

```
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, nd 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 AppCompat 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 to many items on the toolbar to help the screen keep decluttered

The AppCompat 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:

```
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setHasOptionsMenu(true);
}
```

3.1.4 Responding to Menu Selection

Override onOptionsItemSected(...) 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.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case R.id.new_crime:
      .... Do some logic here ....
      return true;   // Return true to say that the selection has
          been handled.
    default:
      return super.onOptionsItemSelected(item);
  }
}
```

3.1.5 Reload/update the menu

```
getActivity().invalidateOptionsMenu();
```

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

3.2 Subtitle

```
private void setSubtitle(String subtitle) {
  AppCompatActivity activity = (AppCompatActivity) getActivity();
  activity.getSupportActionBar().setSubtitle(subtitle);
}
```

- Get the current activity

  (We are using AppCompat 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.

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

3.3.1 How Hierarchical Navigation works

Page 261

4 AppCompat Library

4.1 Requirements

The AppCompat requires that you:

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

5 SQLite Database

5.1 Defining a Schema

```
public class CrimeDbSchema {

  public static final class CrimeTable {
    public static final String NAME = "crimes";

    public static final class Cols {
      public static final String UUID = "uuid";
      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 id 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
         i1) {
17
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).
         getWritableDatabase();
4  }
```

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 ContentVales - 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   ContentValues values = new ContentValues();
3   values.put(CrimeTable.Cols.UUID, crime.getId().toString());
4   values.put(CrimeTable.Cols.TITLE, crime.getTitle());
5   values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
6   values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);
7
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(
           CrimeTable.Cols.UUID));
9      String title = getString(getColumnIndex(CrimeTable.Cols.TITLE))
           ;
10     long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
11     int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));
```

```
12
13      Crime crime = new Crime(UUID.fromString(uuidString));
14      crime.setTitle(title);
15      crime.setDate(new Date(date));
16      crime.setSolved(isSolved != 0);
17
18      return crime;
19    }
20  }
```

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

```
mDatabase.delete(CrimeTable.NAME,
          CrimeTable.Cols.UUID + " = ?",
          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 appropait

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 the be declared within the AppManifest file.

```
<activity android:name=".BrowserActivity">
  <intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:scheme="http" android:host="www.somewhere.com"/>
  </intent-filter>
</activity>
```

6.3 Sending Text

6.3.1 With option of send

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
intent.putExtra(Intent.EXTRA_SUBJECT, getString(R.string.
    crime_report_subject));
intent = Intent.createChooser(intent, getString(R.string.
    send_report));
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.

```
final Intent pickContactIntent = new Intent(Intent.ACTION_PICK,
    ContactsContract.Contacts.CONTENT_URI);
startActivityForResult(pickContactIntent, REQUST_CONTACT);
```

6.4.2 The Receival of Data

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

```
if (data == null) {
  break;
}

Uri contactUri = data.getData();
String[] queryFields = new String[] {
  ContactsContract.Contacts.DISPLAY_NAME
};
Cursor cursor = getActivity().getContentResolver().query(contactUri
    , queryFields, null, null, null);
try {
  if (cursor.getCount() == 0) {
    return;
  }

  cursor.moveToFirst();
  String suspect = cursor.getString(0);
  mCrime.setSuspect(suspect);
  mSuspectButton.setText(suspect);
} finally {
    cursor.close();
}
```

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.

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

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*.

```
PackageManager packageManager = getActivity().getPackageManager();
mPhotoButton = (ImageButton) view.findViewById(R.id.crime_camera);
final Intent captureImageIntent = new Intent(MediaStore.
    ACTION_IMAGE_CAPTURE);
boolean canTakePhoto = (mPhotoFile != null) && (captureImageIntent.
    resolveActivity(packageManager) != null);
mPhotoButton.setEnabled(canTakePhoto);
mPhotoButton.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {
    Uri uri = FileProvider.getUriForFile(getActivity(), "",
        mPhotoFile);
    captureImageIntent.putExtra(MediaStore.EXTRA_OUTPUT, uri);

    List<ResolveInfo> cameraActivities = getActivity().
        getPackageManager().queryIntentActivities(
        captureImageIntent, PackageManager.MATCH_DEFAULT_ONLY);

    for (ResolveInfo activity : cameraActivities) {
      getActivity().grantUriPermission(activity.activityInfo.
          packageName, uri, Intent.FLAG_GRANT_WRITE_URI_PERMISSION)
          ;
    }

    startActivityForResult(captureImageIntent, REQUEST_PHOTO);
  }
});
```

6.5.1 Declaring the Camera Feature

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

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

7 File Storage

Can store files in out 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 access 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 intern, 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.

```
<provider
  android:authorities="com.bgsoftwarestudios.criminalintent.
      fileprovider"
  android:name="android.support.v4.content.FileProvider"
  android:exported="false"
  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 $\rightarrow$ Android resource file.
3. For Resource type select XML.

4. Enter $files$ for the file name.

5. Enter the flowing into the the xml file

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
  <files-path
    name="crime_photos"
    path="."/>
</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:

```
<provider
  android:authorities="com.bgsoftwarestudios.criminalintent.
      fileprovider"
  android:name="android.support.v4.content.FileProvider"
  android:exported="false"
  android:grantUriPermissions="true">
  <meta-data
    android:name="android.support.FILE_PROVIDER_PATHS"
    android:resource="@xml/files"/>
</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.

```
Uri uri = FileProvider.getUriForFile(getActivity(),
    FILE_PROVIDER_AUTHORITY, mPhotoFile);
getActivity().revokeUriPermission(uri, Intent.
    FLAG_GRANT_WRITE_URI_PERMISSION);
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 :
             widthScale;
18       inSampleSize = Math.round(scale);
19     }
20
21     options = new BitmapFactory.Options();
22     options.inSampleSize = inSampleSize;
23
24     // Read in and create final bitmap
25     return BitmapFactory.decodeFile(path, options);
26   }
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 the 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
       activity) {
2    Point size = new Point();
3    activity.getWindowManager().getDefaultDisplay().getSize(size);
4    return getScaledBitmap(path, size.x, size.y);
5  }
```

9 Strings

9.1 Plurals

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

To retrieve/use the string:

```
getResources().getQuantityString(R.plurals.subtitle_plural,
    crimeSize, crimeSize);
```

9.2 Percentages

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

10 Intents

10.1 Starting n activity in a new task

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

11 Two-Pane Master-Detail

This is most commonly used for tablets.
However, you swill commonly require different layout for on phone vs tablet.
To do this, use a 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*.

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

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

See chapter 18

13 Accessibility

13.1 TalkBack

TalkBack if an Android screen reader made by Google.

See Chapter 19.

14 Styles and Themes

14.1 Styles

Declared inside the *styles.xml* file

```
<style name="BeatBoxButton">
  <item name="android:background">@color/dark_blue</item>
</style>
```

14.2 Style Inheritance

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

```
<style name="BeatBoxButton">
  <item name="android:background">@color/dark_blue</item>
</style>

<style name="BeatBoxButton.Strong">
  <item name="android:textStyle">bold</item>
</style>
```

14.3 Themes

Are applied to all objects across the app.

14.3.1 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.

14.3.2 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, AppCompat), use the explicit *parent* = "" attribute.

14.3.3 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 retreivenes the color that the attribute 'colorAccent' in the theme points to.

15 Assets

15.1 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.

15.2 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.

15.3 Accessing Assets

Assets are accessed using the *AssetsManager* class.

15.3.1 Getting an *AssetsManager*

You get get at *AssetsManager* from any context.

15.3.2 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.

15.4 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 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.

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

15.4.2 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(
        sound.getAssetPath());
3   int soundId = soundPool.load(assetFileDescriptor, 1); // Loads a
        file into soundPool for later play back.
4   sound.setSoundId(soundId);
5 }
```

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*.

15.4.3 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.

16 XML Drawables

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

16.1 What is a Drawable

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

16.2 State List Drawables

Is a drawable reasource file that points to other drawable resource files for different states. This file if 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:

```
<selector xmlns:android="http://schemas.android.com/apk/res/android
    ">

  <item android:drawable="@drawable/button_beat_box_pressed"
    android:state_pressed="true"/>
  <item android:drawable="@drawable/button_beat_box_normal"/>

</selector>
```

Then in the first file that is pointed to:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="oval">

  <solid
    android:color="@color/red"/>

</shape>
```

and in the second:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="oval">

  <solid
    android:color="@color/dark_blue"/>

</shape>
```

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

16.3 shape drawables

16.4 layer list drawables

Allows two XML drawables to be combined into one.

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/
    android">

  <item>
    <shape
      android:shape="oval">

      <solid
        android:color="@color/red"/>

    </shape>
  </item>

  <item>
    <shape
      android:shape="oval">

```

```
18        <stroke
19          android:width="4dp"
20          android:color="@color/dark_red"/>
21      </shape>
22    </item>
23
24  </layer-list>
```

17 9-Patch Images

See Page 450

18 AsyncTasks

18.1 *AsyncTasks.cancel(boolean)*

18.1.1 *.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

18.1.2 *.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.

18.2 message queue

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

18.3 Looper

The *looper* is the object that manages a thread's message queue.
Multiple *handlers* can be attached to one *looper*.

18.4 Handler

18.5 Background Thread

18.5.1 Assembling a Background Thread

18.6 *ConcurrentHashMap*

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

19 HTTP

19.1 Make get requests

```
public byte[] getUrlBytes(String urlString) throws IOException {
  URL url = new URL(urlString);
  HttpURLConnection connection = (HttpURLConnection) url.
      openConnection();

  try {
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream
        ();
    InputStream inputStream = connection.getInputStream();

    if (connection.getResponseCode() != HttpURLConnection.HTTP_OK)
        {
      throw new IOException(connection.getResponseMessage() + ":
          with " + urlString);
    }

    int bytesRead = 0;
    byte[] buffer = new byte[1024];
    while ((bytesRead = inputStream.read(buffer)) > 0) {
      outputStream.write(buffer, 0, bytesRead);
    }
    outputStream.close();
    return outputStream.toByteArray();
  } finally {
    connection.disconnect();
  }
}

public String getUrlString(String urlString) throws IOException {
  return new String(getUrlBytes(urlString));
}
```

19.2 building a Url String

```
try {
  String url = Uri.parse("https://www.flickr.com/services/rest/")
    .buildUpon()
    .appendQueryParameter("method", "flickr.photos.getRecent")
    .appendQueryParameter("api_key", API_KEY)
    .appendQueryParameter("format", "json")
    .appendQueryParameter("nojsoncallback", "1")
    .appendQueryParameter("extras", "url_s")
    .build().toString();
  String jsonString = getUrlString(url);
  Log.i(TAG, "fetchItems: Retreived JSON: " + jsonString);
} catch (IOException ioe) {
  Log.e(TAG, "fetchItems: Failed to fetch items", ioe);
}
```

19.3 Parsing JSON

20 *LruCache*

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

20.1 Creating a new Cache

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

of size 40MiB:

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

20.2 Retrieval from the Cache

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

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

20.3 Adding item to cache

```
private void addToCache(String key, Bitmap bitmap) {
  synchronized (thumbnailCache) {
    if (thumbnailCache.get(key) == null) {
      thumbnailCache.put(key, bitmap);
    }
  }
}
```

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.

21 *SearchView*

A *SeachView* is an *action view* class.
It can be embedded into any *view* including a *ToolBar*.

22 SharedPreferences

Should use

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

from the *androidx* library instead of

```
  content...
```

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

```
public class QueryPreferences {

  private static final String TAG = "QueryPreferences";
  private static final String PREF_SEARCH_QUERY = "searchQuery";

  public static String getStoredQuery(Context context) {
    return PreferenceManager.getDefaultSharedPreferences(context)
    .getString(PREF_SEARCH_QUERY, null);
  }

  public static void setStoredQuery(Context context, String query)
      {
    PreferenceManager.getDefaultSharedPreferences(context)
    .edit()
    .putString(PREF_SEARCH_QUERY, query)
    .apply();
  }
}
```

23 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*

23.1 *IntentService*

An *IntentService* is probably the most common service.

23.1.1 Basic Skeletal example

```
public class PollService extends IntentService {
  private static final String TAG = "PollService";

  public static Intent newIntent(@NonNull Context context) {
    return new Intent(context, PollService.class);
  }

  public PollService() {
    super(TAG);
  }

  @Override
  protected void onHandleIntent(Intent intent) {
    Log.i(TAG, "onHandleIntent: Recieved an Intent: " + intent);
  }
}
```

23.2 *commands*

A service's intents and called *commands*.
Each *command* is an instruction for the service to do something.
Since services respond to intents, they must declared in *AndroidManifest/xml*, just like activities do. Like activities

23.3 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.

23.3.1 Checking for background networking availability

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

23.4 *AlarmManager*

AlarmManager is a system service that can send intents for you.
If you used *handler*s 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.

24 Notifications

24.1 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);
```

25 Broadcast Intents

25.1 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.

25.2 Receiving a system Broadcast

25.2.1 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.

25.2.2 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" />
```

```
  </intent-filter>
</receiver>
```

Also requires the permission:

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

25.3 Sending Broadcast Intents

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

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

25.4 *Dynamic Broadcast Receiver*

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

25.4.1 Register

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

```
public abstract class VisibleFragment extends Fragment {

  private static final String TAG = "VisibleFragment";

  @Override
  public void onStart() {
    super.onStart();
    IntentFilter intentFilter = new IntentFilter(PollService.
        ACTION_SHOW_NOTIFICATION);
    getActivity().registerReceiver(onShowNotification, intentFilter
        );
  }

  @Override
  public void onStop() {
    super.onStop();
    getActivity().unregisterReceiver(onShowNotification);
  }

  private BroadcastReceiver onShowNotification = new
      BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
      Toast.makeText(getActivity(), "Got a broadcast: " + intent.
          getAction(), Toast.LENGTH_LONG).show();
    }
  };
}
```

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*

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

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

25.4.2 Un-Register

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

25.5 *IntentFilter*

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

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

26 StrictMode

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

```
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

27 Unit Testing

27.1 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'

27.1.1 *androidTest* vs *test*

andoridTests 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.

*test*s tend to be faster

27.2 Setting Up the Test

27.2.1 Using Mocked Dependencies

27.3 Writing Tests

28 Refactoring Techniques and Tools

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

28.1 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.

29 Building the APK

29.1 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 denisity, containing

only the necessary resources.
See configure-apk-splits

29.1.1 Exception

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