Android

Ben Gavan

July 26, 2019

Contents

1 Dialogs

1.1 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:

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

1.2 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 }
```

1.2.1 Receiving the Data from the intent

```
@Override
public void onActivityResult(int requestCode, int resultCode,
    Intent data) {
  if (resultCode != Activity.RESULT_OK) {
    return;
  }

  if (requestCode == REQUEST_DATE) {
    Date date = (Date) data.getSerializableExtra(DatePickerFragment
        .EXTRA_DATE);
    mCrime.setDate(date);
    mDateButton.setText(mCrime.getDate().toString());
  }
}
```

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)

```
if (resultCode != Activity.RESULT_OK) {
  return;
}
```

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.

```
if (requestCode == REQUEST_DATE) {
  Date date = (Date) data.getSerializableExtra(DatePickerFragment.
      EXTRA_DATE);
  mCrime.setDate(date);
  mDateButton.setText(mCrime.getDate().toString());
}
```

2 The Toolbar

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

2.0.1 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.

2.0.2 Supported by

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

2.1 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.

2.1.1 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:

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

2.1.2 Defining an item

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

The line

```
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.

2.1.3 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);
}
```

2.1.4 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);
  }
}
```

2.1.5 Reload/update the menu

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

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

2.2 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.

2.3 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"/>
```

2.3.1 How Hierarchical Navigation works

Page 261

3 AppCompat Library

3.1 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

4 SQLite Database

4.1 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";
      public static final String DATA = "date";
      public static final String SOLVED = "solved";
    }
  }
}
```

4.2 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.

4.3 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(....)

4.4 Writing to the Database

4.4.1 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 }
```

4.4.2 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 }
```

4.4.3 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.

4.5 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.

4.5.1 Retrieving a Cursor

```
public Cursor queryCrimes(String whereClause, String[] whereArgs) {
  Cursor cursor  = mDatabase.query(
    CrimeTable.NAME,
    null,  // selects all columns
    whereClause,
    whereArgs,
    null, // groupBy
    null,  // having
    null  // orderBy
  );
  return cursor;
}
```

4.5.2 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.

```
String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
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

```
public class CrimeCursorWrapper extends CursorWrapper {

  public CrimeCursorWrapper(Cursor cursor) {
    super(cursor);
  }

  public Crime getCrime() {
    String uuidString = this.getString(this.getColumnIndex(
        CrimeTable.Cols.UUID));
    String title = getString(getColumnIndex(CrimeTable.Cols.TITLE))
        ;
    long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
    int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));

    Crime crime = new Crime(UUID.fromString(uuidString));
    crime.setTitle(title);
    crime.setDate(new Date(date));
    crime.setSolved(isSolved != 0);

    return crime;
  }
}
```

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.

```
public List<Crime> getCrimes() {
  List<Crime> crimes = new ArrayList<>();
  CrimeCursorWrapper cursorWrapper = queryCrimes(null, null);

  try {
    cursorWrapper.moveToFirst();
    while (!cursorWrapper.isAfterLast()) {
      crimes.add(cursorWrapper.getCrime());
      cursorWrapper.moveToNext();
    }
  } finally {
    cursorWrapper.close();
  }

  return crimes;
}
```

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:

```
public Crime getCrime(UUID id) {
  CrimeCursorWrapper cursor = queryCrimes(
    CrimeTable.Cols.UUID + "",
    new String[] { id.toString() }
  );

  try {
    if (cursor.getCount() == 0) {
    return null;
  }

  cursor.moveToFirst();
  return cursor.getCrime();
  } finally {
    cursor.close();
  }
}
```

4.6 Deleting Rows

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

5 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.

5.1 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

6 Strings

6.1 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);
```

7 Refactoring Techniques and Tools

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

7.1 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.