Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Android Persistence using SQLite

Agenda & Goals

- Be aware of the different approaches to data persistence in Android Development
- Be able to work with the **`SQLiteOpenHelper`** & **`SQLiteDatabase`** classes to implement an SQLite database on an Android device (to manage our Donations)
- Be able to work with **`Realm`** to implement a noSQL database on an Android device (again, to manage our Donations)
- Be able to work with **`SharedPreferences`** to manage our Login & Register screens

Data Storage Solutions *

- **Shared Preferences**
  - Store private primitive data in key-value pairs.
- Internal Storage
  - Store private data on the device memory.
- External Storage
  - Store public data on the shared external storage.
- **SQLite Databases**
  - Store structured data in a private database.
- Network Connection
  - Store data on the web with your own network server.

Data Storage Solutions *

❑ Bundle Class

- A mapping from String values to various `Parcelable` types and functionally equivalent to a standard `Map`.
- Does not handle Back button scenario. App restarts from scratch with no saved data in that case.

❑ File

- Use `java.io.*` to read/write data on the device's internal storage.

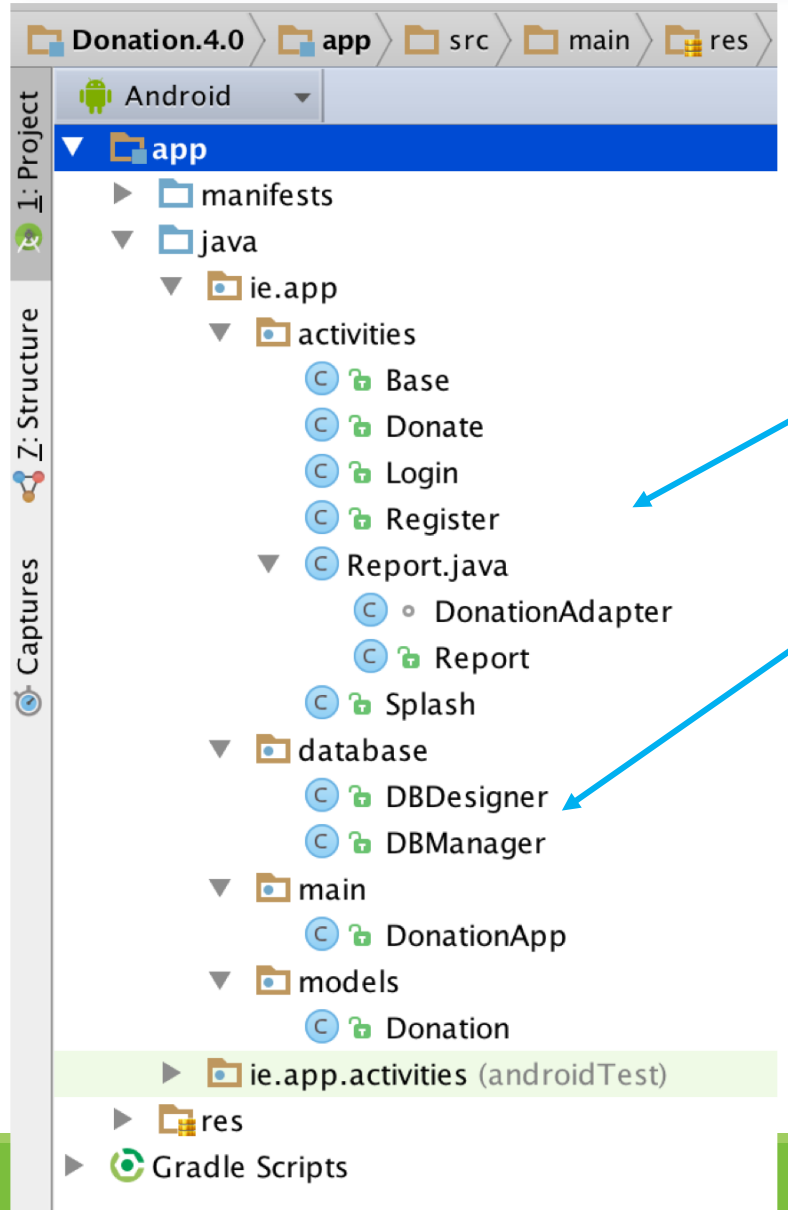
❑ Realm Databases

- Store non-structured data in a private database.

Donation.4.0

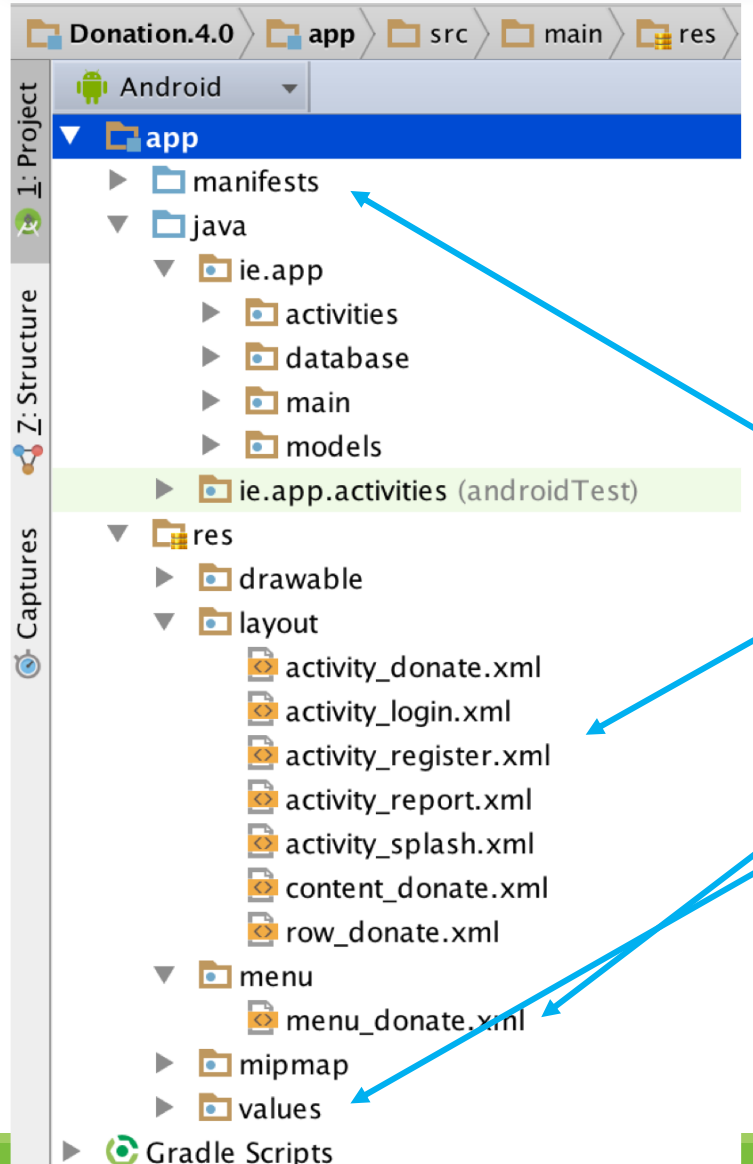
Using an SQLite Database

Donation 4.0 – Project Structure (Classes)



- 9 java source files
  - Our Database classes
- 3 xml layouts
- 1 xml menu
- 6 xml files for resources
- 1 xml 'configuration' file

Donation 4.0 – Project Structure (Resources)



- 9 java source files
  - Our Database classes
- 7 xml layouts
- 1 xml menu
- 6 xml files for resources
- 1 xml 'configuration' file

Idea

- Goal
  - Enhance **Donation.3.0** by managing the Donations in an SQLite Database.
- Approach
  - Implement/extend specific classes to add the database functionality to the app.

Database Programming in Android *

- Android provides full support for `SQLite` databases. Any databases you create will be accessible by name to any class in the application, but not outside the application.
- The recommended method to create a new SQLite database is to create a subclass of `SQLiteOpenHelper` and override the `onCreate()` method, in which you can execute a SQLite command to create tables in the database. For example:

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

Database Programming in Android *

- You can then get an instance of your `SQLiteOpenHelper` implementation using the constructor you've defined. To write to and read from the database, call `getWritableDatabase()` and `getReadableDatabase()`, respectively. These both return a `SQLiteDatabase` object that represents the database and provides methods for `SQLite` operations.
- You can execute `SQLite` queries using the `SQLiteDatabase query()` methods, which accept various query parameters, such as the table to query, the projection, selection, columns, grouping, and others. For complex queries, such as those that require column aliases, you should use `SQLiteQueryBuilder`, which provides several convenient methods for building queries.
- Every SQLite query will return a `Cursor` that points to all the rows found by the query. The Cursor is always the mechanism with which you can navigate results from a database query and read rows and columns.

Database Programming in Android

- With `SQLite`, the database is a simple disk file. All of the data structures making up a relational database - tables, views, indexes, etc. - are within this file
- RDBMS is provided through the api classes so it becomes part of your app
- You can use the SQL you learned in a database module
- You should use DB best practices
  - Normalize data
  - Encapsulate database info in helper or wrapper classes
  - Don’t store files (e.g. images or audio), Instead just store the path string

Step 1 - Define a Schema (and Contract)

- One of the main principles of SQL databases is the schema: a formal declaration of how the database is organized. The schema is reflected in the SQL statements that you use to create your database.
- You may find it helpful to create a companion class, known as a *contract* class, which explicitly specifies the layout of your schema in a systematic and self-documenting way.
- A contract class is a container for constants that define names for URIs, tables, and columns. The contract class allows you to use the same constants across all the other classes in the same package. This lets you change a column name in one place and have it propagate throughout your code.

Donation – DBDesigner *

```
public class DBDesigner extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "donations.db";
    private static final int    DATABASE_VERSION = 1;
    private static final String DATABASE_CREATE_TABLE_DONATION = "create table donations "
            + "(id integer primary key autoincrement,"
            + "amount integer not null,"
            + "method text not null);";

    public DBDesigner(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase database) {
        database.execSQL(DATABASE_CREATE_TABLE_DONATION);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(DBDesigner.class.getName(),
                "Upgrading database from version " + oldVersion + " to "
                        + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS donations");
        onCreate(db);
    }
}
```

Our Table & Column names (SQL) Schema

Creating the Table (or Tables)

Drop the Table (if we change the schema)

Step 2 - Define a Schema (and Contract

- Once you have defined how your database looks, you should implement methods that create and maintain the database and your general CRUD functionality.
- From a design perspective, it can be a good idea to place this functionality in a separate *manager* class, to allow for reuse and SoC (Separation of Concerns)
- This class can act as an APi (of sorts) to allow access to the database from any other class in your application.
- You can refactor your *manager* class internally, without it affecting how other classes call/use the *manager* methods.

Donation – DBManager *

```
public class DBManager {

    private SQLiteDatabase database;
    private DBDesigner dbHelper;

    public DBManager(Context context) { dbHelper = new DBDesigner(context); }

    public void open() throws SQLException {
        database = dbHelper.getWritableDatabase();
    }

    public void close() { database.close(); }

    public void add(Donation d) {
        ContentValues values = new ContentValues();
        values.put("amount", d.amount);
        values.put("method", d.method);

        database.insert("donations", null, values);
    }
```

Our database reference

Returns a reference to the database created from our SQL string

ContentValues are key/value pairs that are used when inserting/updating databases. Each ContentValue object corresponds to one row in a table

Donation – DBManager *

```
public List<Donation> getAll() {
    List<Donation> donations = new ArrayList<>();
    Cursor cursor = database.rawQuery("SELECT * FROM donations", null);
    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
        Donation d = toDonation(cursor);
        donations.add(d);
        cursor.moveToNext();
    }
    cursor.close();
    return donations;
}

private Donation toDonation(Cursor cursor) {
    Donation pojo = new Donation();
    pojo.id = cursor.getInt(0);
    pojo.amount = cursor.getInt(1);
    pojo.method = cursor.getString(2);
    return pojo;
}

public void setTotalDonated(Base base) {
    Cursor c = database.rawQuery("SELECT SUM(amount) FROM donations", null);
    c.moveToFirst();
    if (!c.isAfterLast())
        base.app.totalDonated = c.getInt(0);
}

public void reset() { database.delete("donations", null, null); }
}
```

A Cursor provides random read-write access to the resultset returned by a database query

This method 'converts' a Cursor object into a Donation Object

You can execute standard SQL if you like?

Other Cursor Functions

- moveToPrevious
- getCount
- getColumnIndexOrThrow
- getColumnName
- getColumnNames
- moveToPosition
- getPosition

Donation.4.0

Using the Application Object

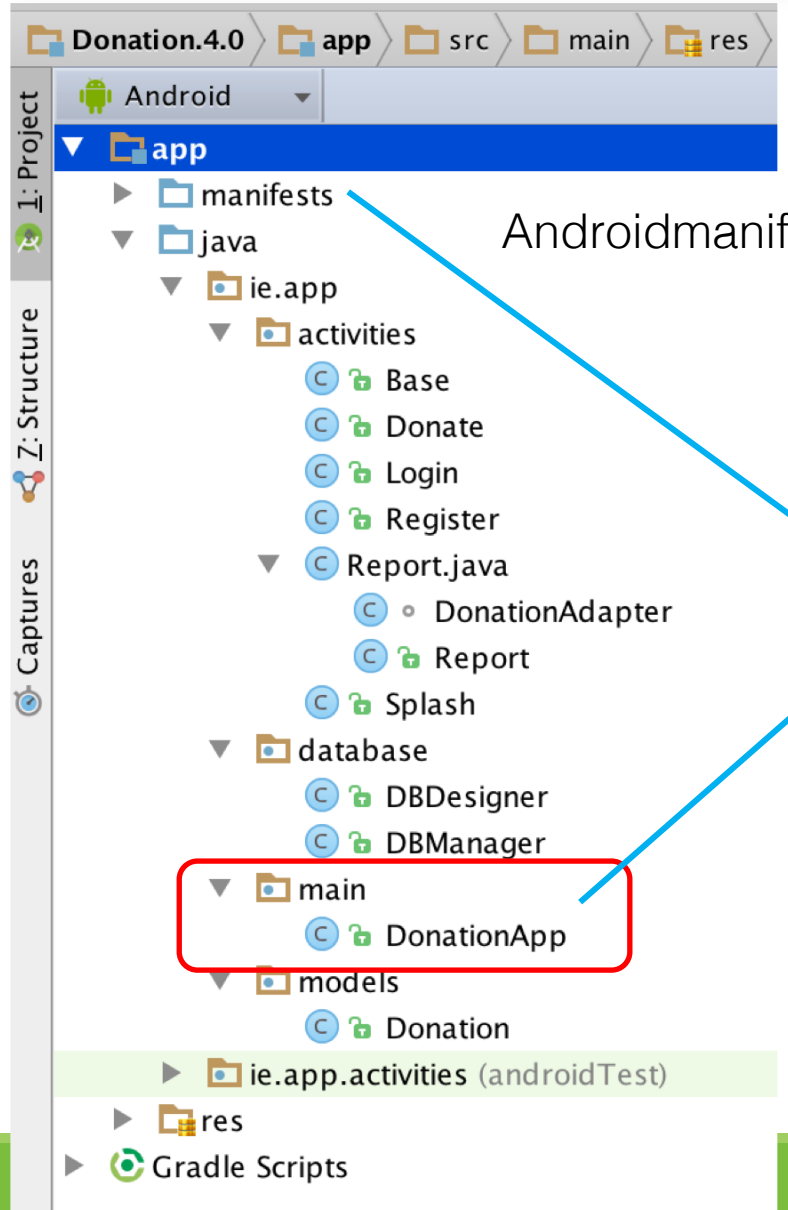
Maintaining Global Application State

- Sometimes you want to store data, like global variables which need to be accessed from multiple Activities – sometimes everywhere within the application. In this case, the Application object will help you.
- Activities come and go based on user interaction
- Application objects can be a useful ‘anchor’ for an android app
- You can use it to hold information shared by all activities

Application Object Callbacks

- onConfigurationChanged( ) Called by the system when the device configuration changes while your component is running.
- onCreate( ) Called when the application is starting, before any other application objects have been created.
- onLowMemory( ) This is called when the overall system is running low on memory, and would like actively running processes to tighten their belts.
- onTerminate( ) This method is for use in emulated process environments. It will never be called on a production Android device, where processes are removed by simply killing them; no user code (including this callback) is executed when doing so.

The Application Object *



Androidmanifest.xml

```
import android.app.Application;

public class DonationApp extends Application
{
   @Override
   public void onCreate()
   {
      super.onCreate();
      Log.v("Donation", "Donation App Started");
   }
}
```

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Donation.4.0"
    android:supportsRtl="true"
    android:theme="@style/AppTheme"
    android:name="ie.app.main.DonationApp">
```

Donation 4.0 – DonationApp Application Object *

```
public class DonationApp extends Application
{
    public final int       target       = 10000;
    public int             totalDonated = 0;
    //public List <Donation> donations    = new ArrayList<Donation>();
    public DBManager dbManager;


    public boolean newDonation(Donation donation)
    {
        boolean targetAchieved = totalDonated > target;
        if (!targetAchieved) {
            dbManager.add(donation);
            totalDonated += donation.amount;
        }
        else
            Toast.makeText(this, "Target Exceeded!", Toast.LENGTH_SHORT).show();

        return targetAchieved;
    }

    @Override
    public void onCreate()
    {
        super.onCreate();
        Log.v("Donate", "Donation App Started");
        dbManager = new DBManager(this);
        Log.v("Donate", "Database Created");

    }
}
```

Our Database Reference

Adding a Donation

Refactor existing Activities/Classes

- In order to make full use of our Database and Application object we need to refactor the classes in the project.
- This will form part of the Practical Lab (Lab 5) but we'll have a quick look now at some of the refactoring that needs to be done.
- We also add in a new Menu option to 'Reset' the database once the target has been reached and keep track of the current total donated if the app is shut down are restarted.

Donation 4.0 – **Base** (Refactored, extract) *

```
public class Base extends AppCompatActivity {

    public DonationApp app;

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

        app = (DonationApp) getApplication();

        app.dbManager.open();
        app.dbManager.setTotalDonated(this);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        app.dbManager.close();
    }
```

Our DonationApp reference

'Binding' to our Application Object

Invoking the Application wide methods

Donation 4.0 – **Base** (Refactored, extract) *

```
@Override
public boolean onPrepareOptionsMenu (Menu menu){
    super.onPrepareOptionsMenu(menu);
    MenuItem report = menu.findItem(R.id.menuReport);
    MenuItem donate = menu.findItem(R.id.menuDonate);
    MenuItem reset = menu.findItem(R.id.menuReset);

    if(app.dbManager.getAll().isEmpty())
    {
        report.setEnabled(false);
        reset.setEnabled(false);
    }
    else {
        report.setEnabled(true);
        reset.setEnabled(true);
    }
    if(this instanceof Donate){
        donate.setVisible(false);
        if(!app.dbManager.getAll().isEmpty())
        {
            report.setVisible(true);
            reset.setEnabled(true);
        }
    }
    else {
        report.setVisible(false);
        donate.setVisible(true);
        reset.setVisible(false);
    }
```

Our new 'Reset' menu option

Checking the database

Donation 4.0 – **Donate** (Refactored) *

```
public void donateButtonPressed (View view)
{
    String method = paymentMethod.getCheckedRadioButtonId() == R.id.PayPal ? "PayPal" : "Direct";
    int donatedAmount =  amountPicker.getValue();
    if (donatedAmount == 0)
    {
        String text = amountText.getText().toString();
        if (!text.equals(""))
            donatedAmount = Integer.parseInt(text);
    }
    if (donatedAmount > 0)
    {
        app.newDonation(new Donation(donatedAmount, method));
        progressBar.setProgress(app.totalDonated);
        String totalDonatedStr = "$" + app.totalDonated;
        amountTotal.setText(totalDonatedStr);
    }
}

@Override
public void reset(MenuItem item)
{
    app.dbManager.reset();
    app.totalDonated = 0;
    amountTotal.setText("$" + app.totalDonated);
}
```

Invoking the Application wide methods

Donation 4.0 – DonationAdapter *

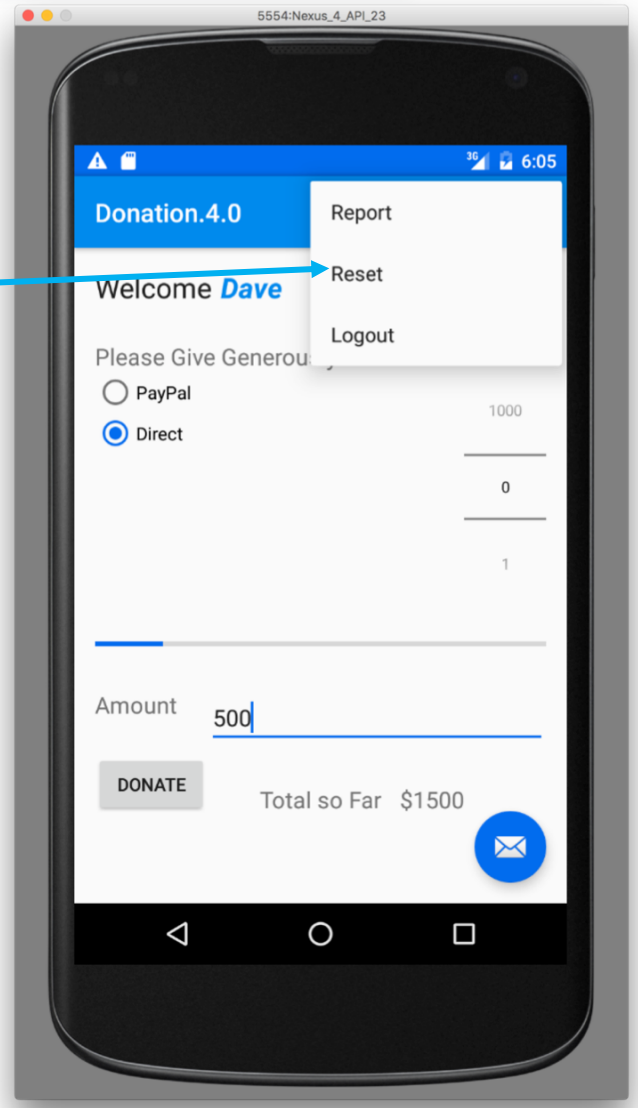
```
class DonationAdapter extends ArrayAdapter<Donation>
{
    private Context context;
    public List<Donation> donations;

    public DonationAdapter(Context context, List<Donation> donations)
    {
        super(context, R.layout.row_donate, donations);
        this.context   = context;
        this.donations = donations;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent)
    {
        LayoutInflater inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        View     view       = inflater.inflate(R.layout.row_donate, parent, false);
        Donation donation   = donations.get(position);
        TextView amountView = (TextView) view.findViewById(R.id.row_amount);
        TextView methodView = (TextView) view.findViewById(R.id.row_method);

        amountView.setText("$" + donation.amount);
        methodView.setText(donation.method);

        view.setId(donation.id); // setting the id of the 'row' to the id of the donation

        return view;
    }

    @Override
```

Donation 4.0 – Report

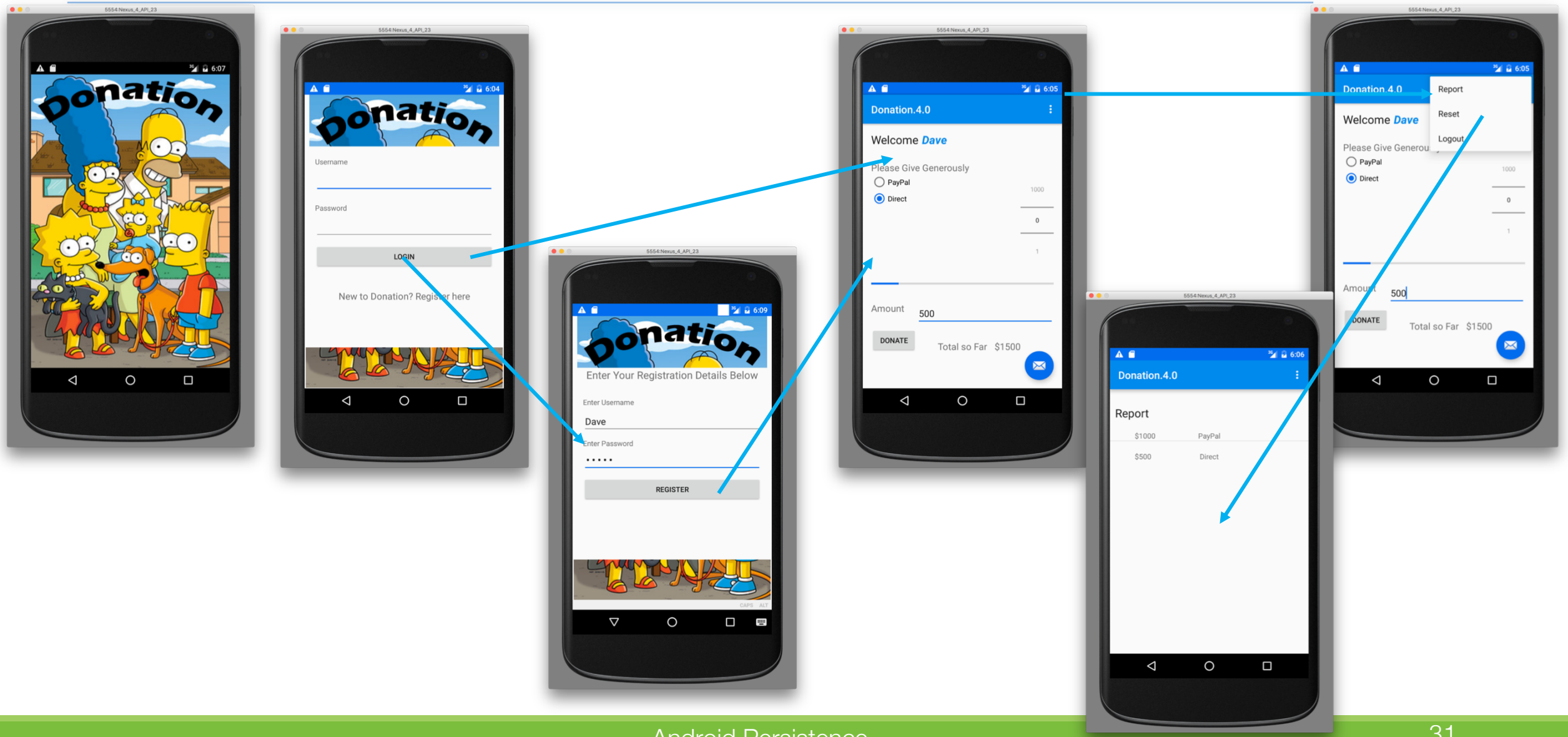
```
public class Report extends Base implements OnItemClickListener
{
    ListView listView;
    DonationAdapter adapter;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_report);

        listView = (ListView) findViewById(R.id.reportList);
        adapter = new DonationAdapter(this, app.dbManager.getAll());
        listView.setAdapter(adapter);
        listView.setOnItemClickListener(this);
    }

    @Override
    public void onItemClick(AdapterView<?> arg0, View arg1, int pos, long id) {

        Toast.makeText(this, "You Selected Row [ " + pos + "]\n" +
                "For Donation Data [ " + adapter.donations.get(pos) + "]\n " +
                "With ID of [" + id + "]", Toast.LENGTH_LONG).show();

    }
}
```

Donation 4.0 – 'Reset' Menu Item

```
<item
   android:id="@+id/menuReset"
   android:orderInCategory="100"
   android:showAsAction="never"
   android:title="@string/menuReset"
   android:onClick="reset"/>
```



Ultimate Case Study – Donation 4.0



Questions?