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

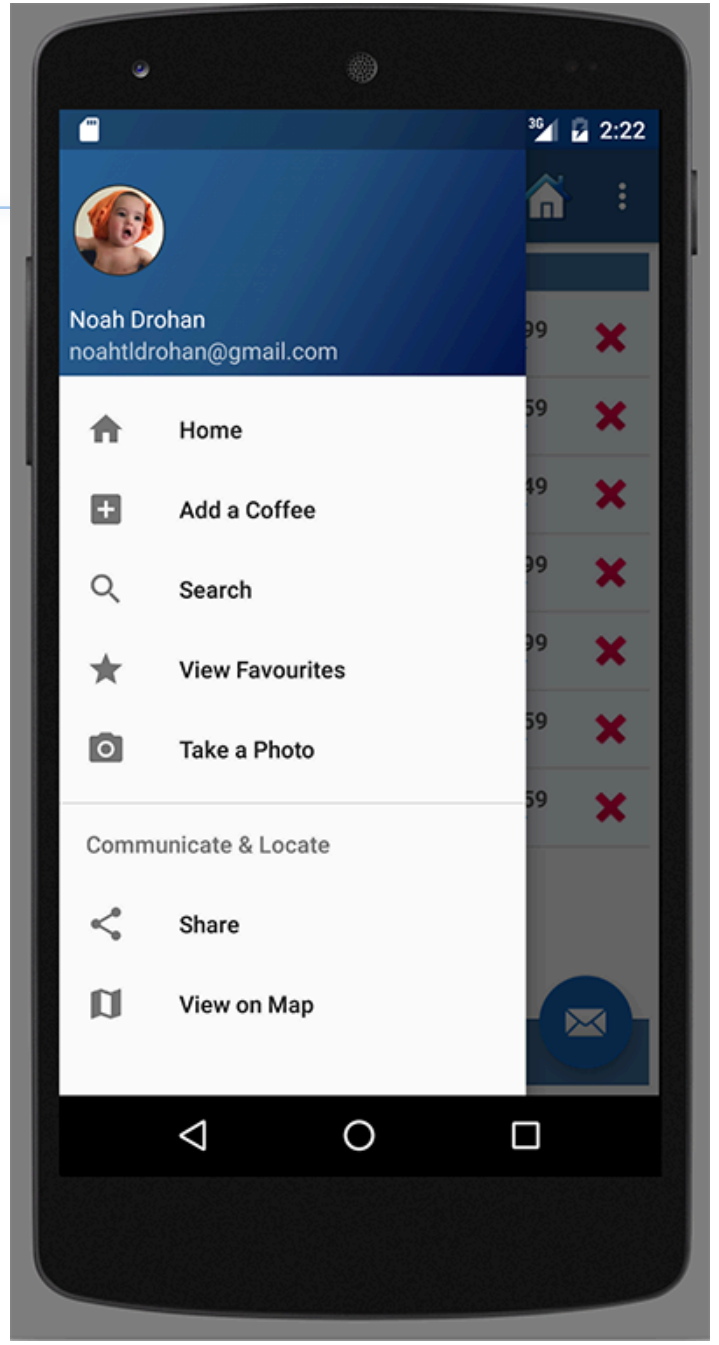
User Interface Design & Development – Part 2

Goals of this Section

- Be able to create and use some more different widgets (views) and features such as **`Spinners`** and **`Filters`**
- Share data between Activities using the **`Application`** object
- Understand how to develop and reuse **`Fragments`** in a multi-screen app
- Be able to create and use a **`NavigationDrawer`** to implement more effective navigation within an app (**CoffeeMate 4.0+**)

Case Study

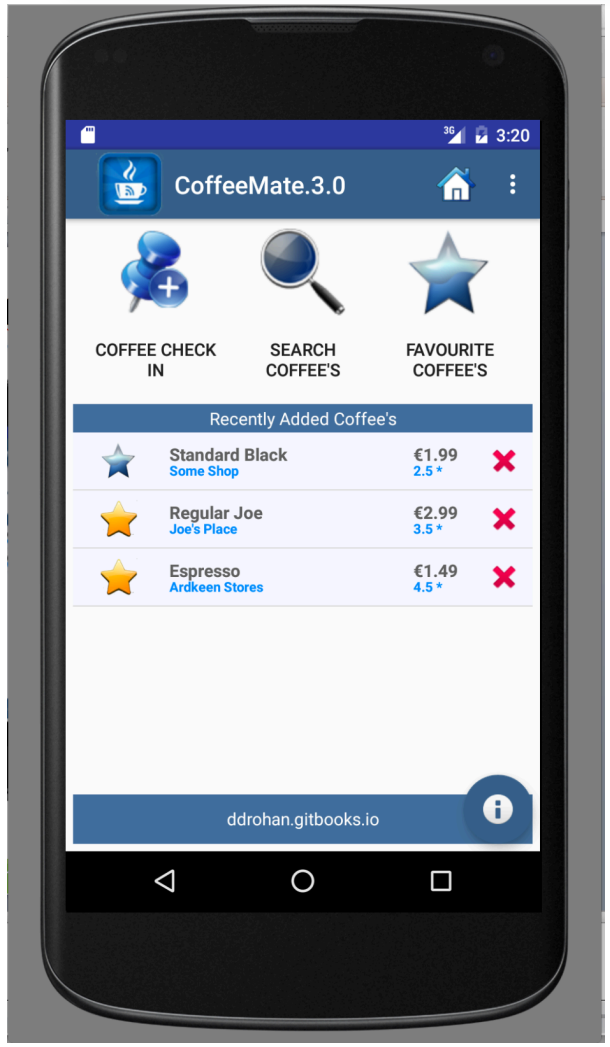
- *CoffeeMate* – an Android App to keep track of your Coffees, their details, and which ones you like the best (your favourites)
- App Features with Google+ Sign-In
  - List all your Coffees
  - View specific Coffee details
  - Filter Coffees by Name and Type
  - Delete a Coffee
  - List all your Favourite Coffees
  - View Coffees on a Map

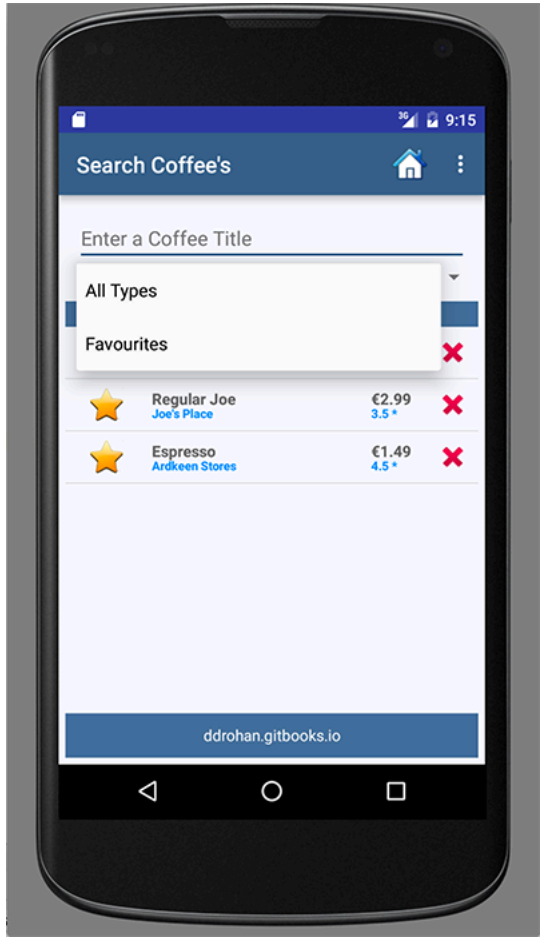


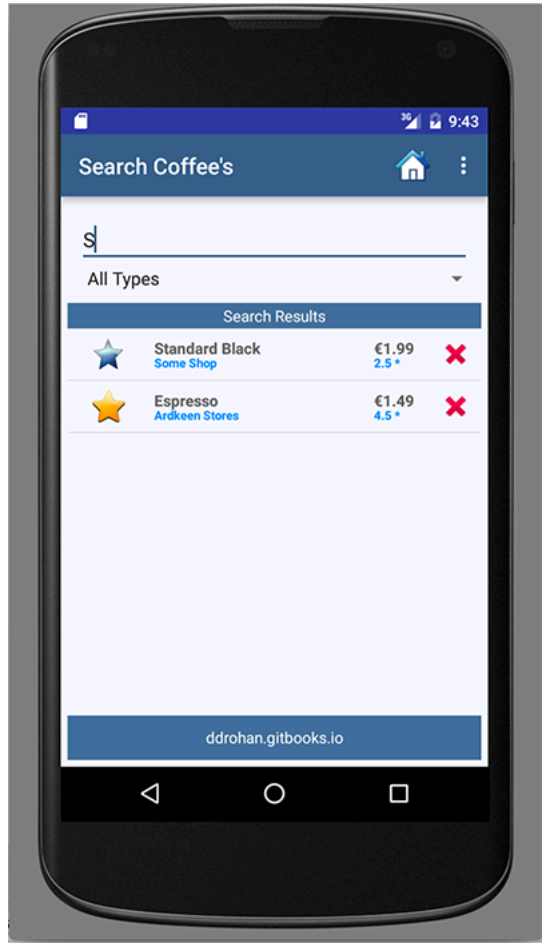
CoffeeMate 3.0

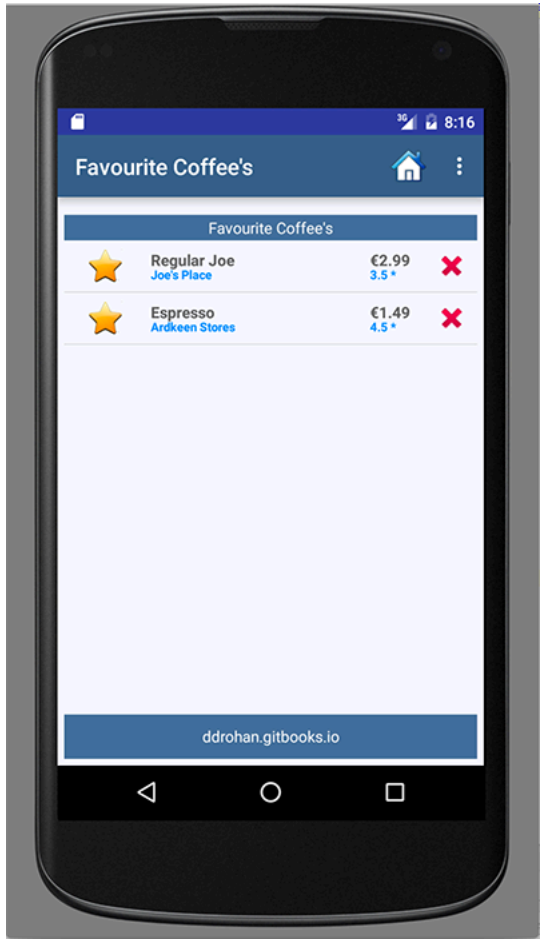
Using Spinners and Filters

CoffeeMate 3.0



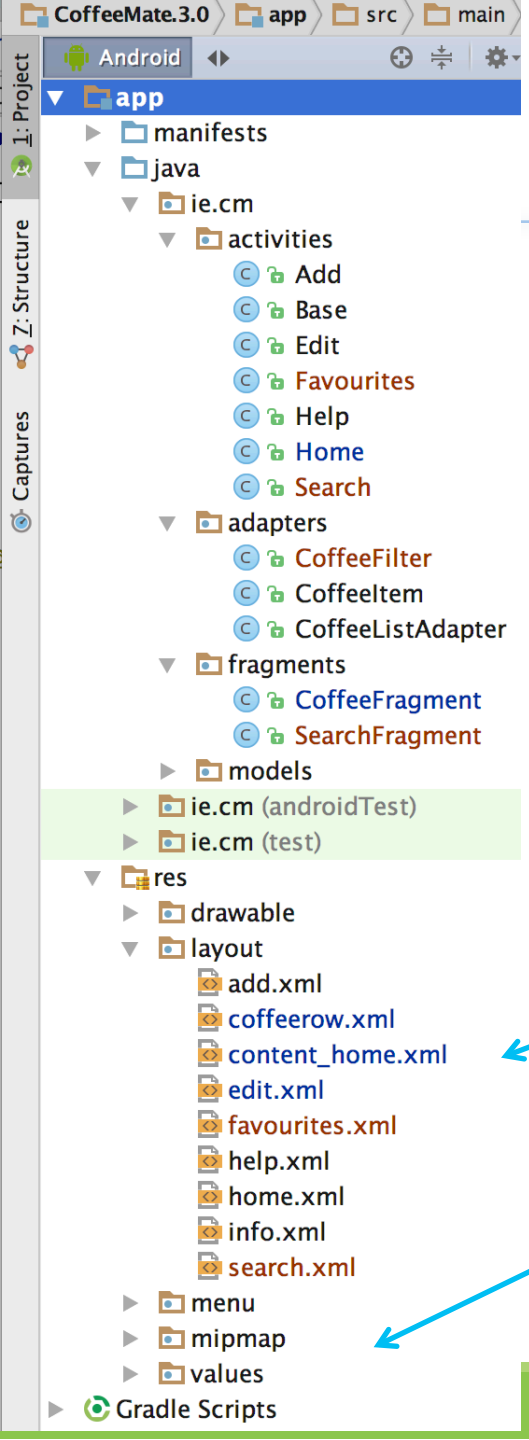






Still no Persistence in this Version

CoffeeMate 3.0



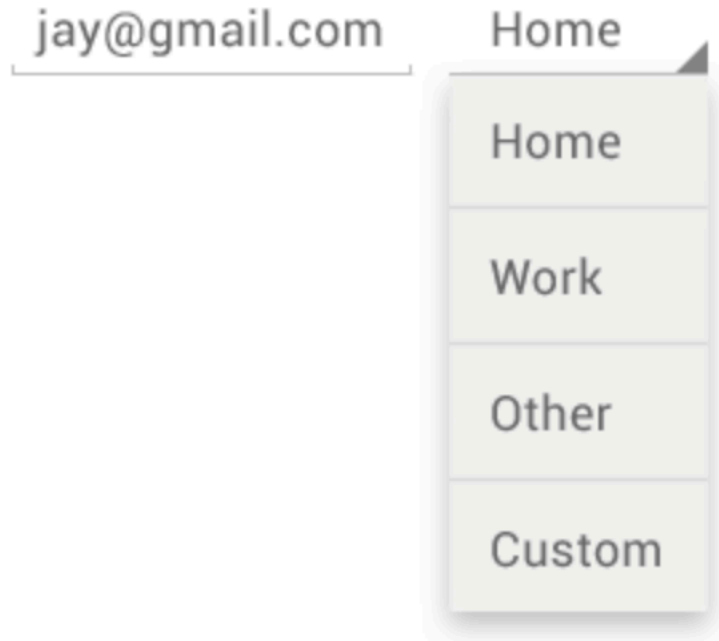
- 4 new java source files
- 2 new xml layouts
- 1 new xml file for resources (specifically the Spinner widget, arrays.xml)

CoffeeMate 3.0

Using Spinners

Overview - Spinners

- Spinners provide a quick way to select one value from a set.
- In the default state, a spinner shows its currently selected value.
- Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.



Overview - Spinners

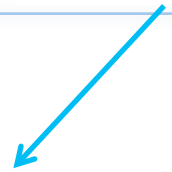
- You can add a spinner to your layout with the **`Spinner`** object. You should usually do so in your XML layout with a <Spinner> element. For example:

```
<Spinner
    android:id="@+id/searchCoffeeTypeSpinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:prompt="Choose a Type of Coffee" />
```

- To populate the spinner with a list of choices, you then need to specify a **`SpinnerAdapter`** in your Activity or Fragment source code (next slide).

Populate the Spinner with User Choices

```
<string-array name="coffeeTypes">
    <item>All Types</item>
    <item>Favourites</item>
</string-array>
```



```
ArrayAdapter<CharSequence> spinnerAdapter = ArrayAdapter
        .createFromResource(activity, R.array.coffeeTypes,
                            android.R.layout.simple_spinner_item);
```

- Then, bind to the **`Spinner`** widget and set its Adapter (and Listener) to display the options to the user.

```
spinnerAdapter
        .setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

Spinner spinner = ((Spinner) activity.findViewById(R.id.searchCoffeeTypeSpinner));
spinner.setAdapter(spinnerAdapter);
spinner.setOnItemSelectedListener(this);
```

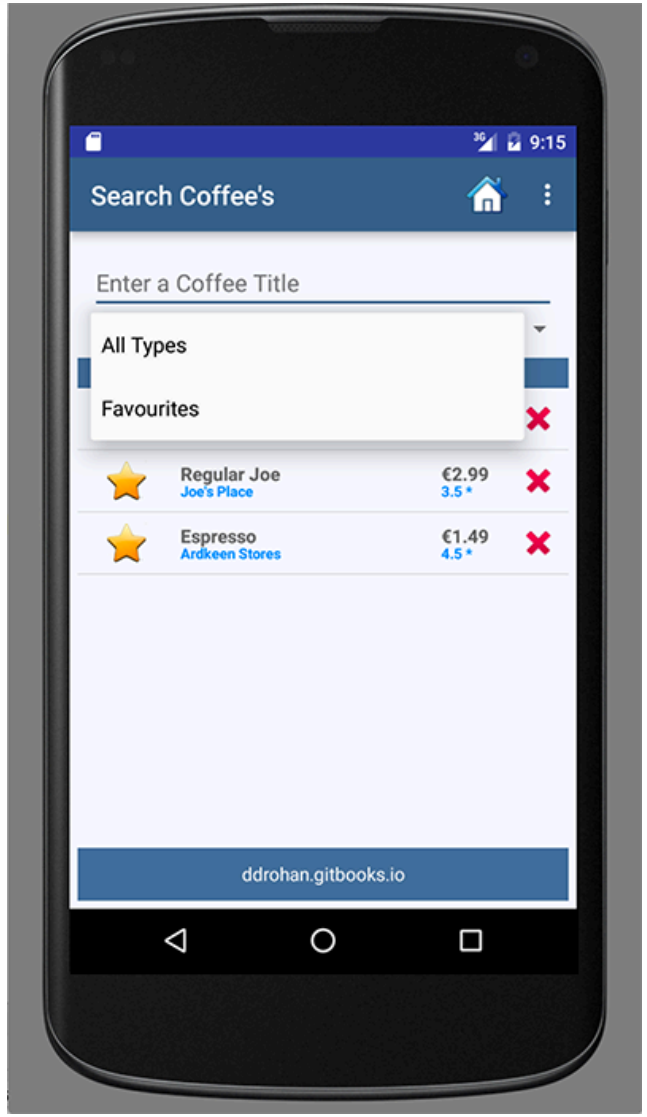
Populate the Spinner with User Choices

```
<string-array name="coffeeTypes">
        <item>All Types</item>
        <item>Favourites</item>
    </string-array>
```

This is the data we use to populate our spinner widget

Key classes

- Spinner
- SpinnerAdapter
- AdapterView.OnItemSelectedListener



CoffeeMate 3.0

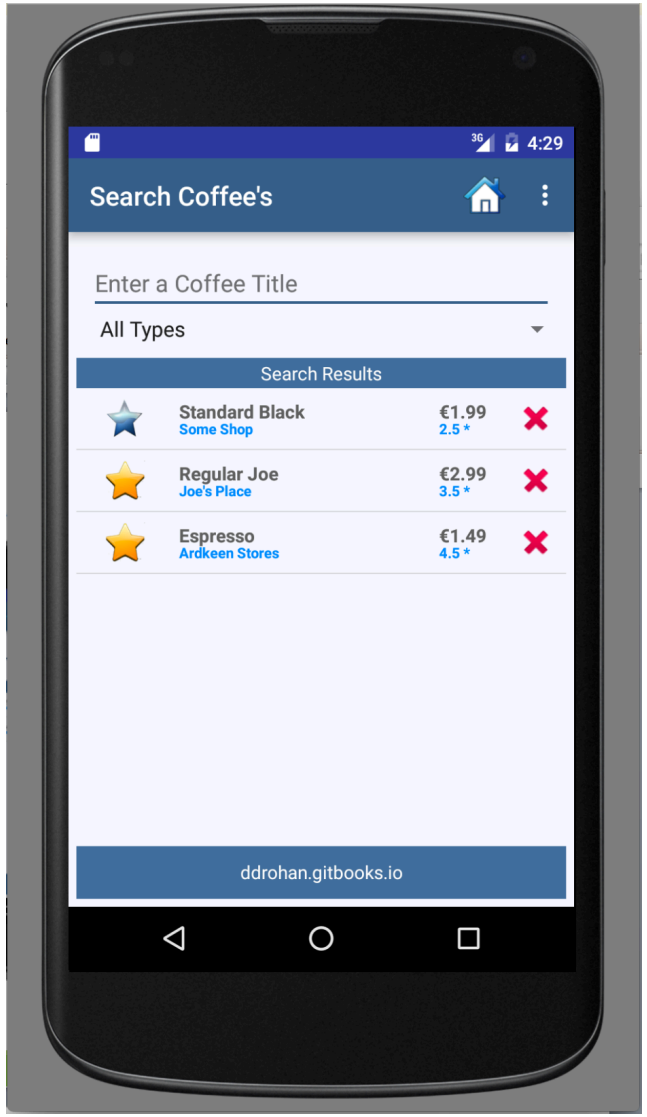
Code Highlights (1)

Search *

VERY similar to our **Home** Activity

```
public class Search extends Base {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.search);
    }

    @Override
    protected void onResume() {
        super.onResume();

        coffeeFragment = SearchFragment.newInstance(); //get a new Fragment instance
        getFragmentManager()
                .beginTransaction()
                .replace(R.id.fragment_layout, coffeeFragment)
                .commit(); // add/replace in the current activity
    }
}
```

Only difference with 'Home' – we'll cover this Fragment in more detail in the labs

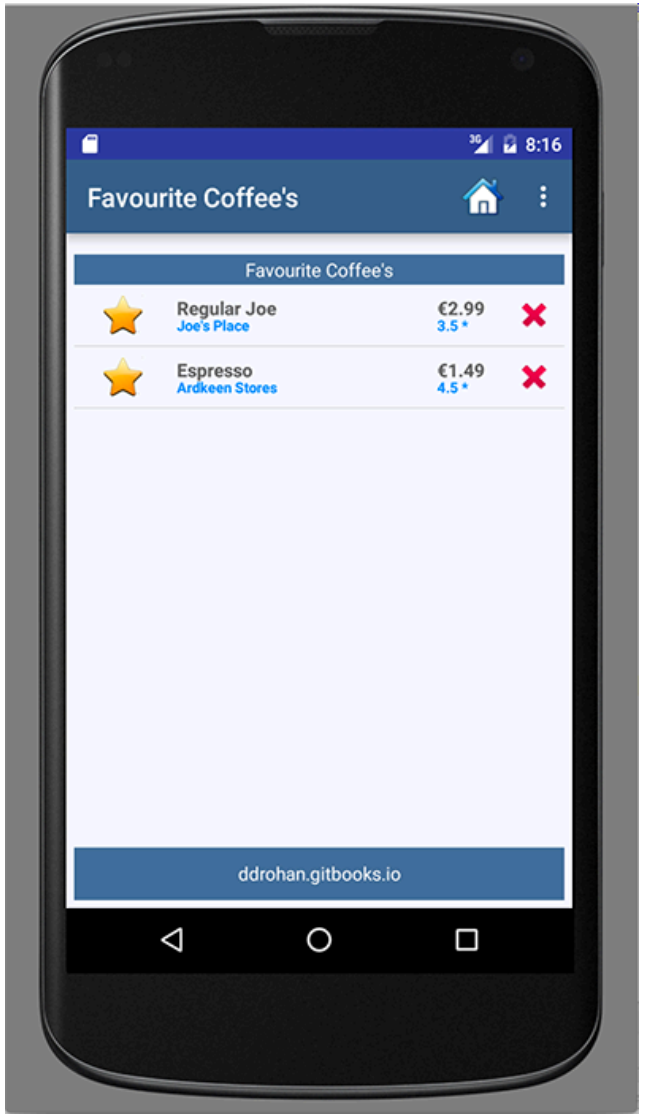


Favourites *

EXACTLY similar to our **Home** Activity

```
public class Favourites extends Base {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.favourites);
    }

    @Override
    protected void onResume() {
        super.onResume();

        coffeeFragment = CoffeeFragment.newInstance(); //get a new Fragment instance
        getFragmentManager()
                .beginTransaction()
                .replace(R.id.fragment_layout, coffeeFragment)
                .commit(); // add/replace in the current activity
    }
}
```

Don't even need to change the Fragment



CoffeeMate 3.0

Using Filters

Filtering & Sorting

- ListView supports filtering of elements via its adapter.
- For example the ArrayAdapter class implements the Filterable interface and contains a default filter implementation called ArrayFilter as an inner class.
- This default implementation allows you to filter based on a String, via

```
youradapter.getFilter().filter(searchString)
```

- Typically you might want to add an EditText field to your layout and attach a TextChangeListener to it. (as with our example)

Filtering & Sorting

- Because we're using a Custom Adapter (our nice rows ☺) and a Custom object (a Coffee) the default implementation isn't sufficient for our needs.
- Our approach is to
  - create a Custom Filter (**`CoffeeFilter`**)
  - maintain a reference to in in our Fragment (**`CoffeeFragment`**)
  - tell the filter what and how to filter the data (our **`Coffee`** object)
- Our **`CoffeeFilter`** has two abstract methods we need to implement
  - **`FilterResults performFiltering(CharSequence constraint)`** : invoked in worker thread, that has the task to filter the results according to the constraint
  - **`void publishResults(CharSequence constraint,FilterResults results)`** : that has the task to show the result set created by performingFiltering method
- So let's have a look…

CoffeeMate 3.0

Code Highlights (2)

CoffeeFragment – Filtering *

```
public class CoffeeFragment  extends ListFragment implements  OnClickListener
{
  protected          Base                activity;
  protected static   CoffeeListAdapter   listAdapter;
  protected          ListView            listView;
  protected          CoffeeFilter        coffeeFilter;
```

Declare a reference to our CoffeeFilter in our Fragment
(so we can filter our 'Favourites')

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

  listAdapter = new CoffeeListAdapter(activity, this, Base.coffeeList);
  coffeeFilter = new CoffeeFilter(Base.coffeeList,"all",listAdapter);

  if (activity instanceof Favourites) {
    coffeeFilter.setFilter("favourites"); // Set the filter text field from 'all' to 'favourites'
    coffeeFilter.filter(null); // Filter the data, but don't use any prefix
    listAdapter.notifyDataSetChanged(); // Update the adapter
  }
  setListAdapter (listAdapter);
}
```

If the associated Activity is 'Favourites', then filter on "favourites"

CoffeeFragment – Filtering after Multiple Deletes *

```
public class CoffeeFragment  extends ListFragment implements  OnClickListener
{
  protected          Base                activity;
  protected static   CoffeeListAdapter   listAdapter;
  protected          ListView            listView;
  protected          CoffeeFilter        coffeeFilter;


private void deleteCoffees(ActionMode actionMode)
{
  for (int i = listAdapter.getCount() - 1; i >= 0; i--)
  {
    if (listView.isItemChecked(i))
      Base.coffeeList.remove(listAdapter.getItem(i));
  }
  actionMode.finish();

  if (activity instanceof Favourites) {
    coffeeFilter.setFilter("favourites");
    coffeeFilter.filter(null);
  }

  listAdapter.notifyDataSetChanged();
}
```

We need to filter again after deleting multiple coffees to get our remaining 'Favourites'.

SearchFragment – Filtering after Multiple Deletes *

```
public class SearchFragment extends CoffeeFragment
        implements AdapterView.OnItemSelectedListener, TextWatcher {

    String selected;
```

```
@Override
public void deleteCoffees(ActionMode actionMode) {
    super.deleteCoffees(actionMode);
    checkSelected(selected);
}
```

We Override the existing 'deleteCoffees' to add some extra functionality

We need to filter again after deleting multiple coffees to get our remaining 'Favourites' and/ or any filtered coffees on text. We'll have a closer look at this method next

CoffeeFragment – Helper Method *

```
private void checkSelected(String selected)
{
    if (selected != null) {
        if (selected.equals("All Types")) {
            coffeeFilter.setFilter("all");
        } else if (selected.equals("Favourites")) {
            coffeeFilter.setFilter("favourites");
        }

        String filterText = ((EditText)activity
                .findViewById(R.id.searchCoffeeNameEditText))
                .getText().toString();

        if(filterText.length() > 0)
            coffeeFilter.filter(filterText);
        else
            coffeeFilter.filter("");
    }
}
```

Filtering again (after deleting multiple coffees) to get our remaining 'Favourites', if any.

Binding to the EditText and filtering again (after deleting multiple coffees) to get our remaining coffees matching certain text entered, if any.

CoffeeFilter (1)

```
public class CoffeeFilter extends Filter {
    private List<Coffee>        originalCoffeeList;
    private String              filterText;
    private CoffeeListAdapter   adapter;

    public CoffeeFilter(List<Coffee> originalCoffeeList, String filterText,
            CoffeeListAdapter adapter) {
        super();
        this.originalCoffeeList = originalCoffeeList;
        this.filterText = filterText;
        this.adapter = adapter;
    }

    public void setFilter(String filterText) {
        this.filterText = filterText;
    }
```

A reference to our adapter so we can update it directly

Setting the text to filter on

CoffeeFilter (2) *

Invoked in a worker thread to filter the data according to the prefix

```
@Override
protected FilterResults performFiltering(CharSequence prefix) {
    FilterResults results = new FilterResults();

    if (originalCoffeeList == null) {
        originalCoffeeList = new ArrayList<Coffee>();
    }
    if (prefix == null || prefix.length() == 0) {
        List<Coffee> newCoffees = new ArrayList<~>();
        if (filterText.equals("all")) {
            results.values = originalCoffeeList;
            results.count = originalCoffeeList.size();
        } else {
            if (filterText.equals("favourites")) {
                for (Coffee c : originalCoffeeList)
                    if (c.favourite)
                        newCoffees.add(c);
            }
            results.values = newCoffees;
            results.count = newCoffees.size();
        }
    } else {
        String prefixString = prefix.toString().toLowerCase();
        final ArrayList<Coffee> newCoffees = new ArrayList<~>();

        for (Coffee c : originalCoffeeList) {
            final String itemName = c.name.toLowerCase();
            if (itemName.contains(prefixString)) {
                if (filterText.equals("all")) {
                    newCoffees.add(c);
                } else if (c.favourite) {
                    newCoffees.add(c);
                }}}
        results.values = newCoffees;
        results.count = newCoffees.size();
    }
    return results;
}
```

Filtering on Spinner Selection (*no text entered*) – Triggered by

```
@Override
public void onItemSelected(AdapterView<?> parent, Vie
    String selected = parent.getItemAtPosition(positi

    if (selected != null) {
        if (selected.equals("All Types")) {
            coffeeFilter.setFilter("all");
        } else if (selected.equals("Favourites")) {
            coffeeFilter.setFilter("favourites");
        }
        coffeeFilter.filter("");
    }
}
```

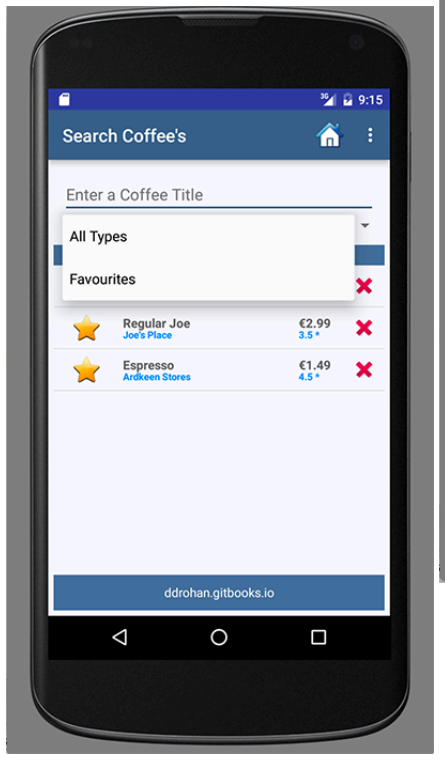
Filtering on *Text* – Triggered by

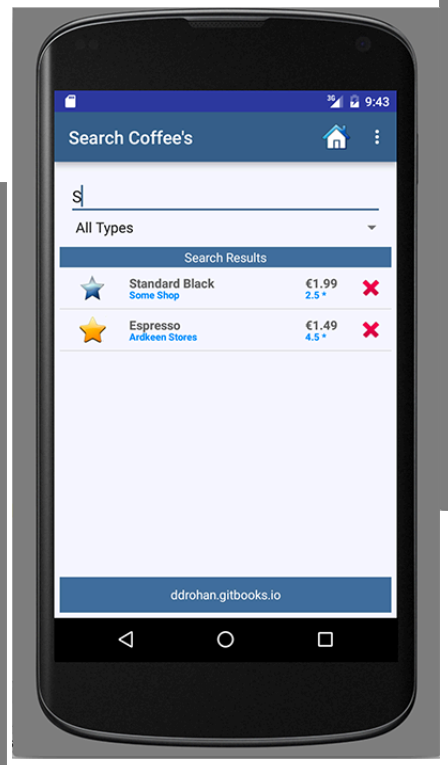
```
@Override
public void onTextChanged(CharSequence s, int sta
    coffeeFilter.filter(s);
}
```

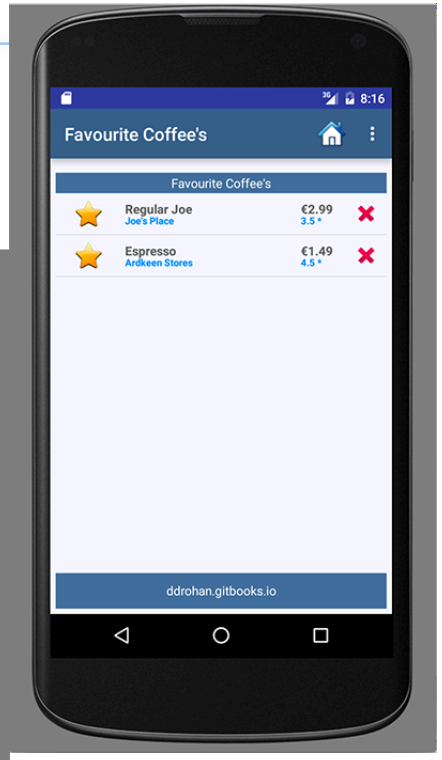
CoffeeFilter (3) *

Invoked in the UI thread to publish the filtering results on the main UI thread (usually the user interface)

```
@Override
protected void publishResults(CharSequence prefix, FilterResults results) {

    adapter.coffeeList = (ArrayList<Coffee>) results.values;

    if (results.count >= 0)
        adapter.notifyDataSetChanged();
    else {
        adapter.notifyDataSetInvalidated();
        adapter.coffeeList = originalCoffeeList;
    }
}
```







CoffeeMate 3.0

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.

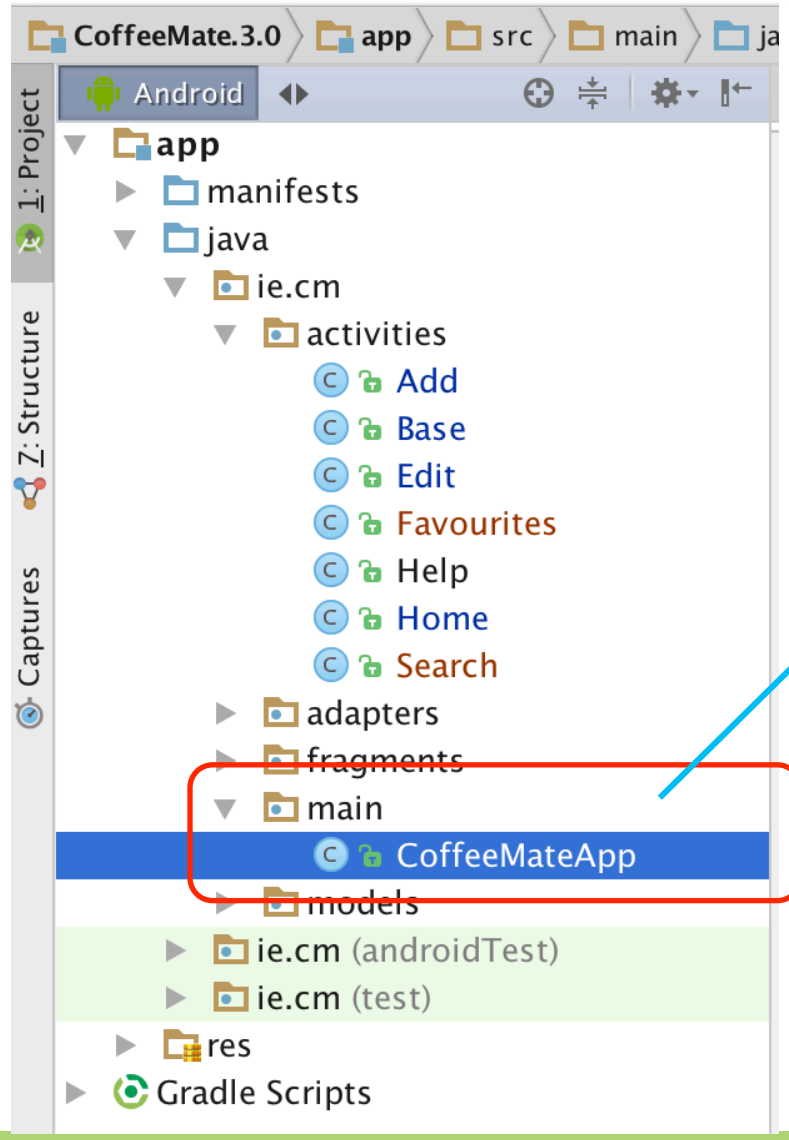
Refactor existing Activities/Classes

- In order to make full use of our Application object we need to refactor some of the classes in the project.
- This will form part of the Practical Lab (Lab 4) but we'll have a quick look now at some of the refactoring that needs to be done to both include, and make use of, our Application object.

CoffeeMate 3.0

Code Highlights (3)

The Application Object *




```
public class CoffeeMateApp extends Application
{
    public List <Coffee>  coffeeList = new ArrayList<Coffee>();

    @Override
    public void onCreate()
    {
        super.onCreate();
        Log.v("coffeemate", "CoffeeMate App Started");
    }
}
```

Androidmanifest.xml

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="CoffeeMate.3.0"
    android:supportsRtl="true"
    android:theme="@style/AppTheme"
    android:name="ie.cm.main.CoffeeMateApp">
```

CoffeeMate 3.0 – code extracts *

```
public class Base extends AppCompatActivity {

    public    CoffeeMateApp              app;
    protected Bundle                     activityInfo; // Used for pe
    protected CoffeeFragment             coffeeFragment; // How we'll

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        app = (CoffeeMateApp) getApplication();
    }
```

Our CoffeeMateApp reference

Binding to our Application Object

```
    if ((coffeeName.length() > 0) && (coffeeShop.length() > 0)
            && (price.length() > 0)) {
        Coffee c = new Coffee(coffeeName, coffeeShop, ratingValue,
                coffeePrice, false);

        app.coffeeList.add(c);
        goToActivity(this,Home.class, null);
    } else
```

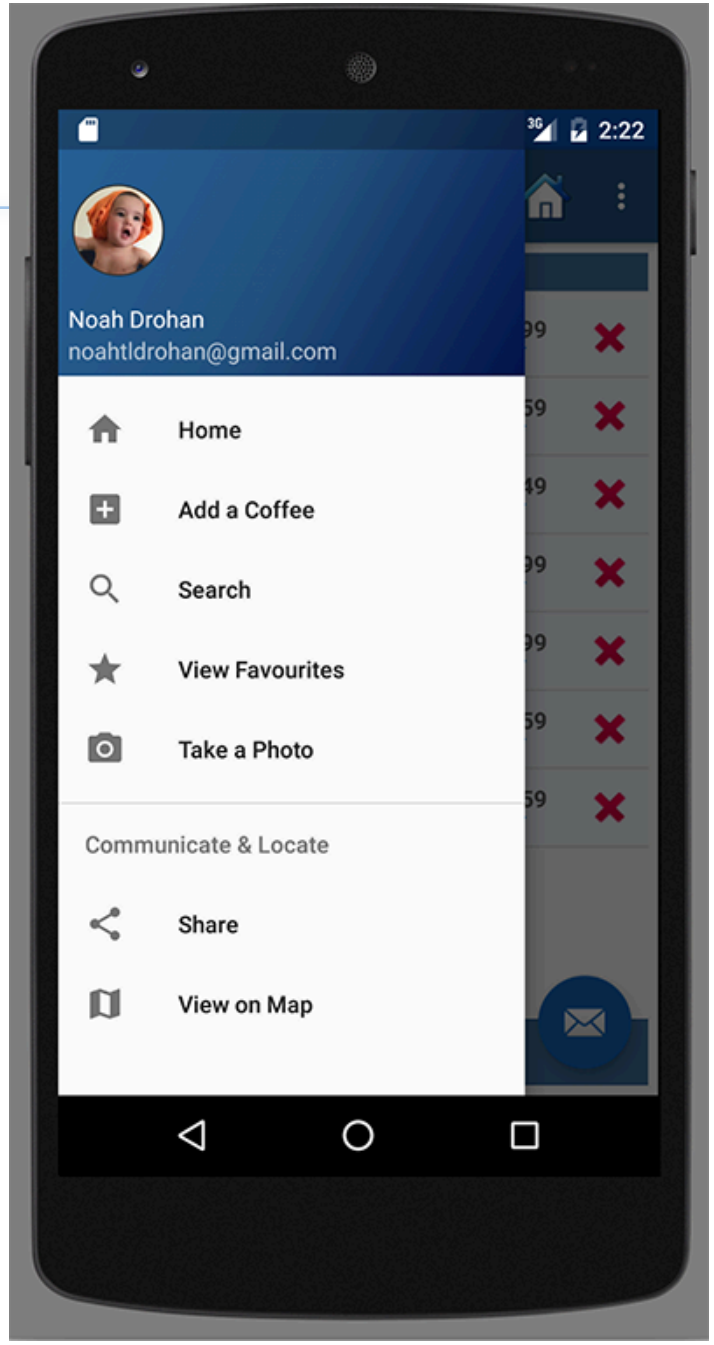
Adding a Coffee to our coffeeList via the Application Object

CoffeeMate 4.0+

Using
The
Navigation Drawer

Navigation Drawer Overview

- https://developer.android.com/training/implementing-navigation/nav-drawer.html
- The navigation drawer is a panel that displays the app's main navigation options on the left edge of the screen. It is hidden most of the time, but is revealed when the user swipes a finger from the left edge of the screen or, while at the top level of the app, the user touches the app icon in the action bar.



Navigation Drawer Overview

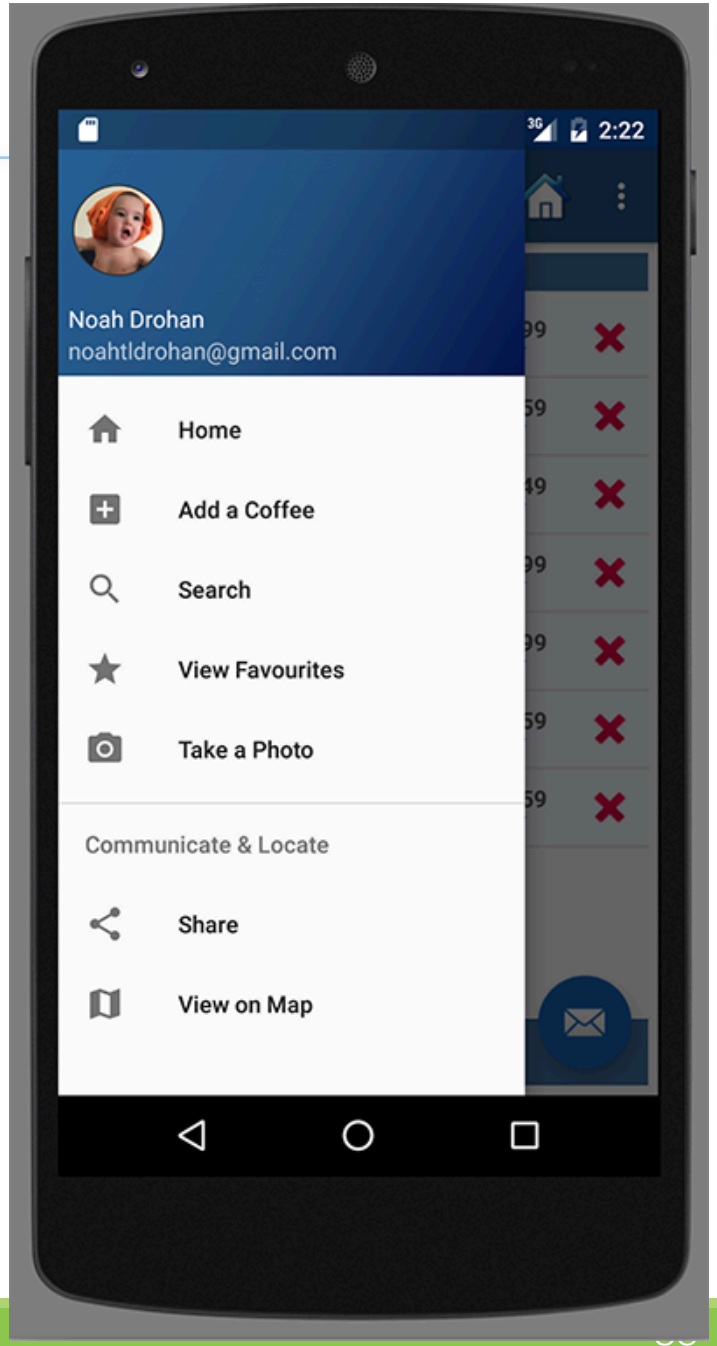
- Android Studio does a lot of the heavy lifting for you, but generally the following steps are necessary to add a Navigation Drawer to your app
  - *Create drawer layout*
  - *Bind to navigation drawer layout*
  - *Handle navigation drawer click and*
  - *Update content based on user selection*

Overview - *Create Drawer Layout*

- For creating a navigation drawer, first we need to declare the drawer layout in your main activity where you want to show the navigation drawer.
- You add **`android.support.v4.widget.DrawerLayout`** as the root view of your activity layout.
- As already mentioned, Android Studio does a lot of this for you so it's more about understanding how it all pieces together to allow you to modify as necessary.
- We'll use **`CoffeeMate`** as the example to illustrate…

Overview - *Create Drawer Layout* *

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:openDrawer="start">

    <include
        layout="@layout/app_bar_home"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:fitsSystemWindows="true"
        app:headerLayout="@layout/nav_header_home"
        app:menu="@menu/activity_home_drawer" />

</android.support.v4.widget.DrawerLayout>
```



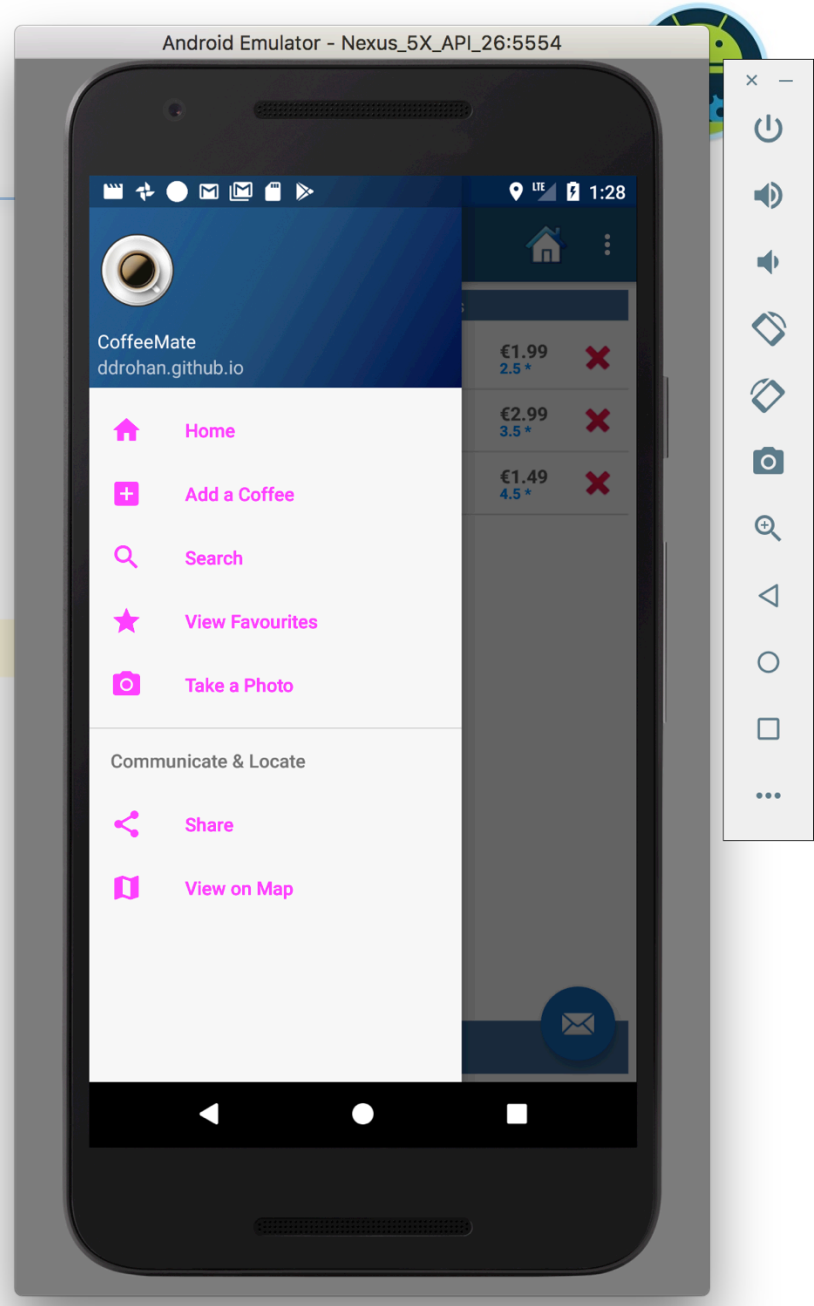
Overview - *Create Drawer Layout* *

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:openDrawer="start">

    <include
        layout="@layout/app_bar_home"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:fitsSystemWindows="true"
        app:headerLayout="@layout/nav_header_home"
        app:menu="@menu/activity_home_drawer" />

</android.support.v4.widget.DrawerLayout>
```

- *activity_home.xml* contains the Navigation **Header** (*nav_header_home*) AND the Navigation Drawer **Menu** (*activity_home_drawer*) inside a **`NavigationView`**.
- *activity_home includes app_bar_home* which will display our content
- Also, note the 'ids' of the widgets (for later on)

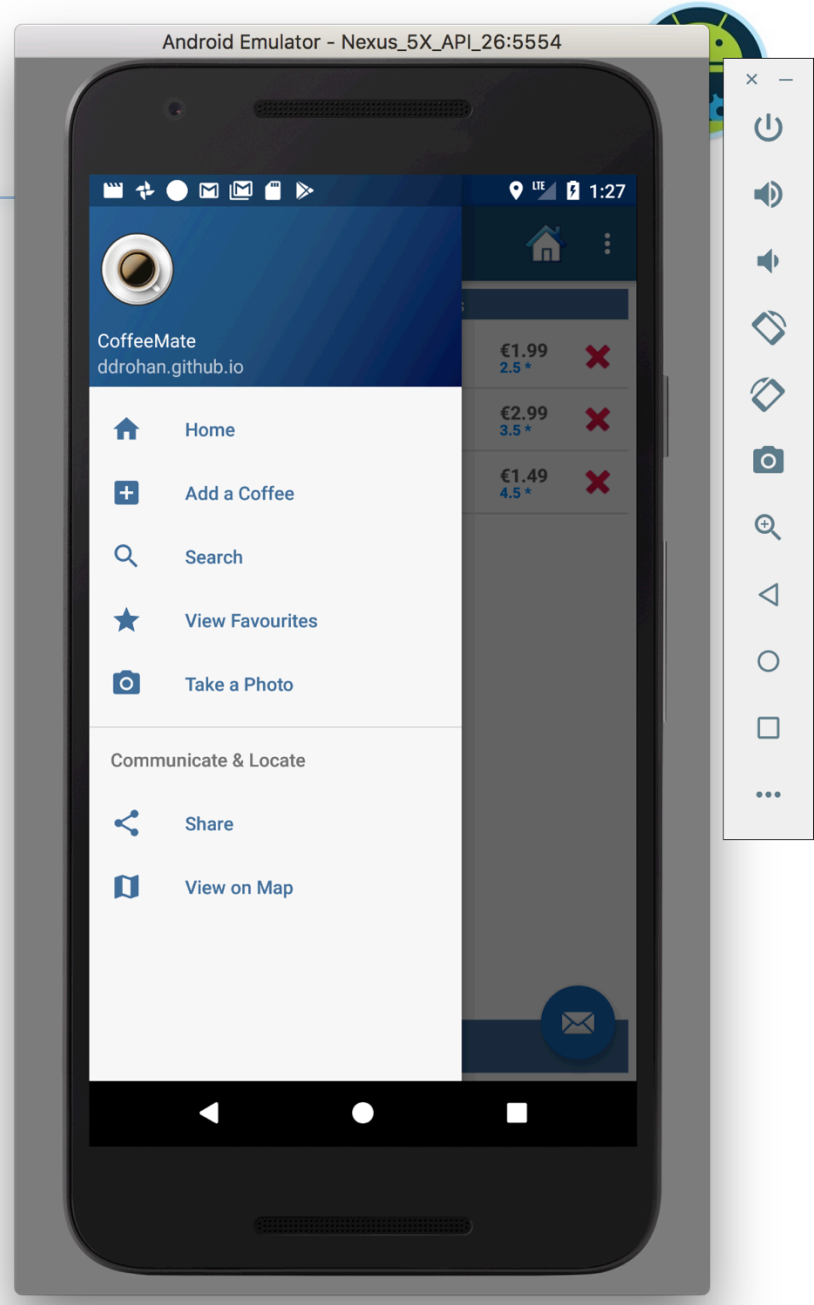
UPDATE

```
16      <android.support.design.widget.NavigationView
17          android:id="@+id/nav_view"
18          android:layout_width="wrap_content"
19          android:layout_height="match_parent"
20          android:layout_gravity="start"
21          android:fitsSystemWindows="true"
22          app:headerLayout="@layout/nav_header_home"
23          app:menu="@menu/activity_home_drawer" />
24
```



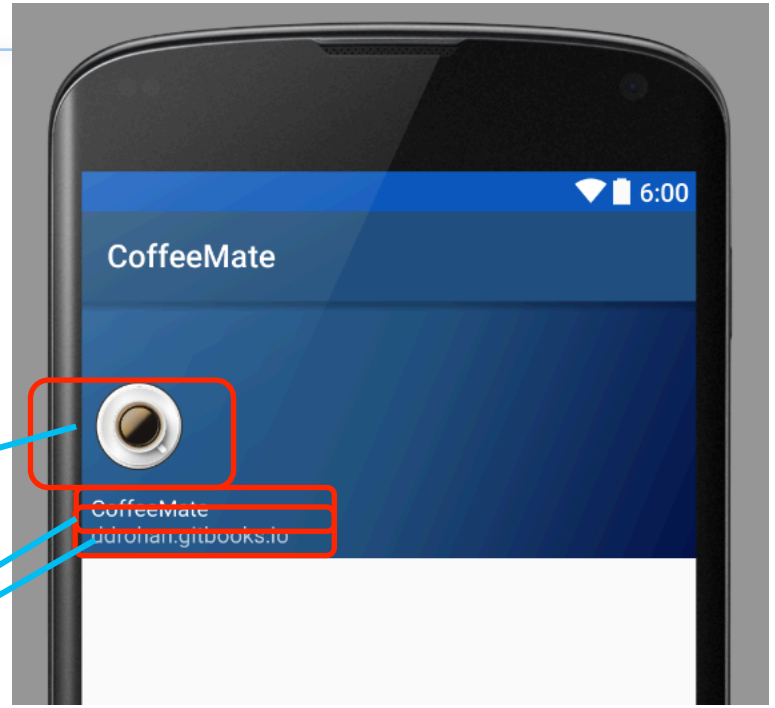
UPDATE

```
15
16      <android.support.design.widget.NavigationView
17          android:id="@+id/nav_view"
18          android:layout_width="wrap_content"
19          android:layout_height="match_parent"
20          android:layout_gravity="start"
21          android:fitsSystemWindows="true"
22          app:headerLayout="@layout/nav_header_home"
23          app:itemTextColor="@color/headerBGColor"
24          app:itemIconTint="@color/headerBGColor"
25          app:menu="@menu/activity_home_drawer" />
26
```



Overview – *nav_header_home* *

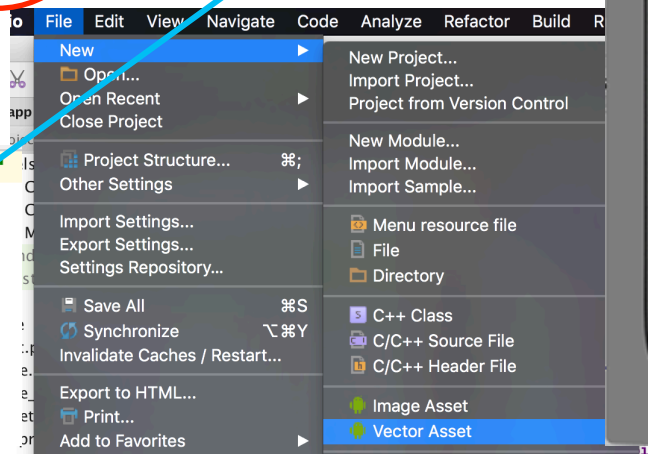
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="160dp"
    android:background="@drawable/side_nav_bar"
    android:gravity="bottom"
    android:orientation="vertical"
    android:paddingBottom="6dp"
    android:paddingLeft="6dp"
    android:paddingRight="6dp"
    android:paddingTop="6dp"
    android:theme="@style/ThemeOverlay.AppCompat.Dark">

    <!-- https://github.com/vinc3m1/RoundedImageView -->
    <com.makeramen.roundedimageview.RoundedImageView...>
    <TextView
        android:id="@+id/googlename"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingTop="16dp"
        android:text="CoffeeMate"
        android:textAppearance="@style/TextAppearance.AppCompat.Body1" />

    <TextView...>

</LinearLayout>
```

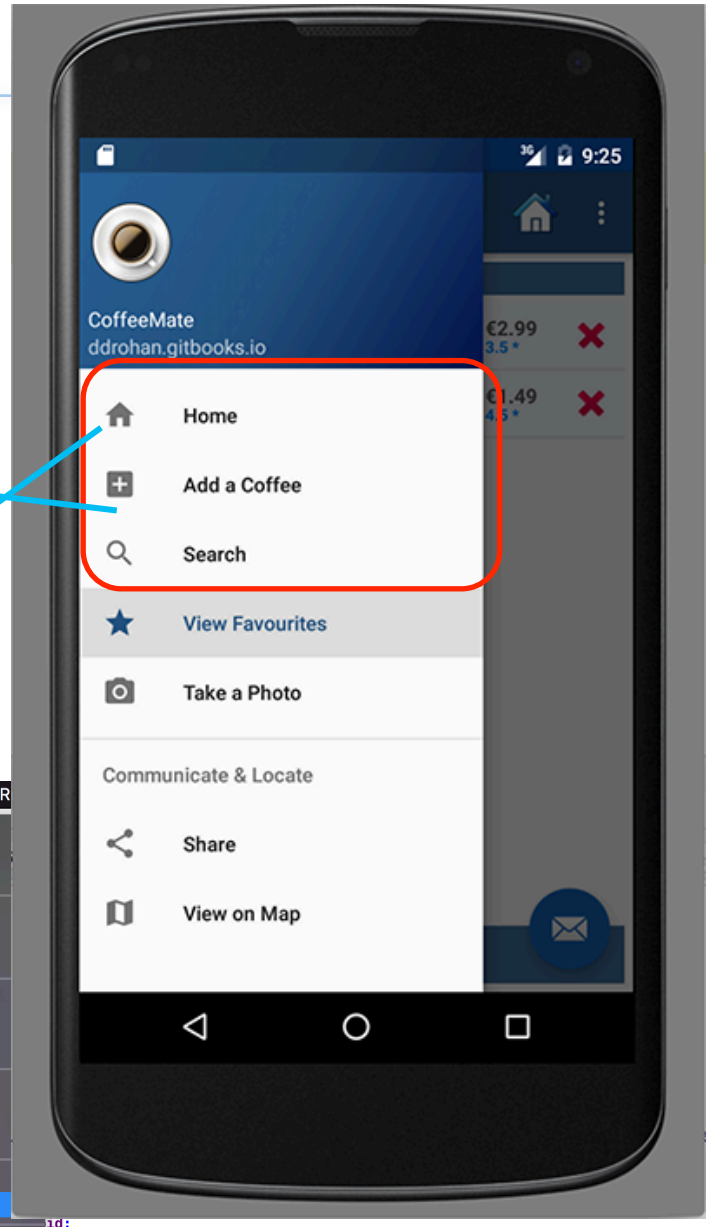


Overview – *activity_home_drawer **

```
<menu xmlns:android="http://schemas.android.com/apk/res/andr
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_home"
            android:icon="@drawable/ic_menu_home"
            android:title="Home" />
        <item
            android:id="@+id/nav_add"
            android:icon="@drawable/ic_menu_addcoffee"
            android:title="Add a Coffee" />
        <item
            android:id="@+id/nav_search"
            android:icon="@drawable/ic_menu_search"
            android:title="Search" />
        <item
```

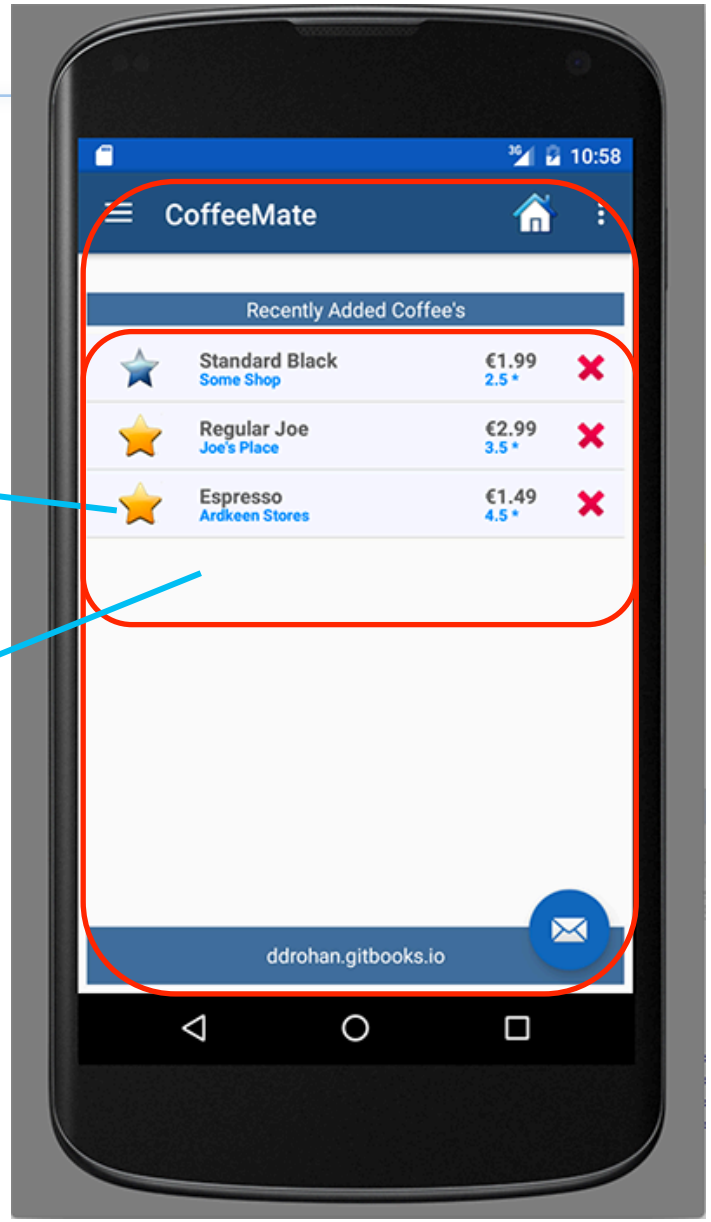
```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
        android:width="24dp"
        android:height="24dp"
        android:viewportWidth="24.0"
        android:viewportHeight="24.0">
    <path
        android:fillColor="#FF000000"
        android:pathData="M10,20v-6h4v6h5v-8h3L12,3 2,12h3v8z"/>
</vector>
```





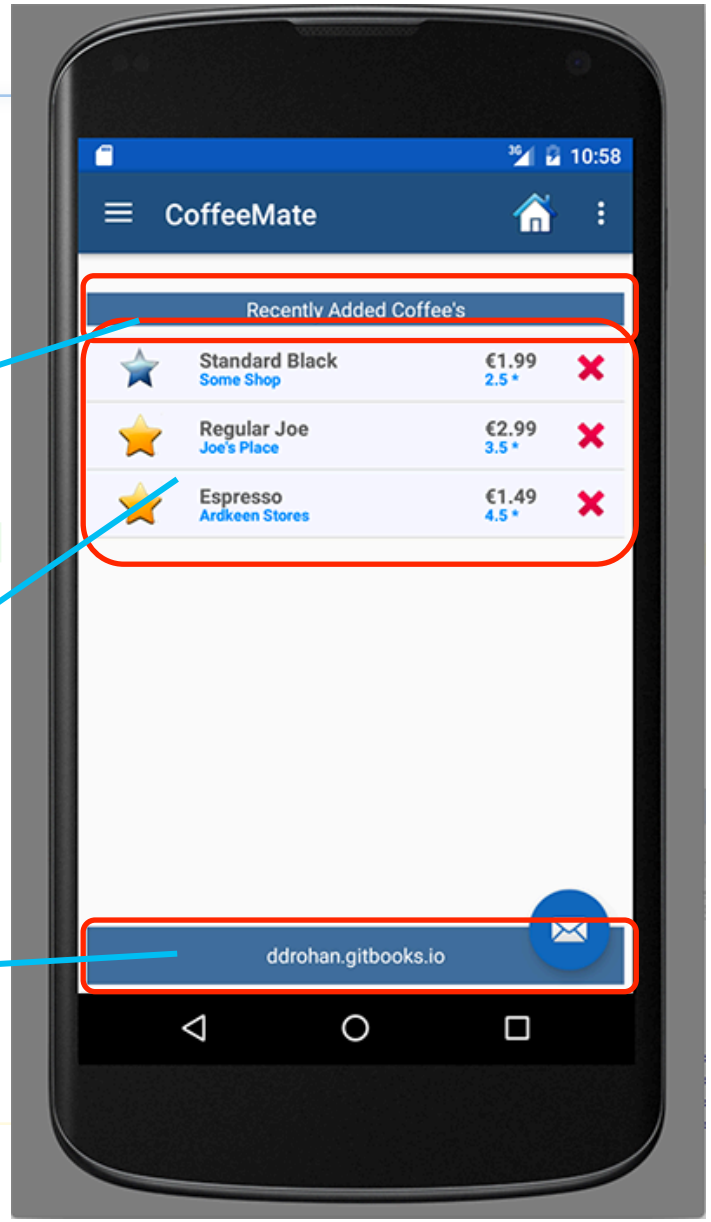
Overview – *app_bar_home* *

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context=".activities.Home">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar...>

    </android.support.design.widget.AppBarLayout>

    <include layout="@layout/content_home" />

    <android.support.design.widget.FloatingActionButton...>
</android.support.design.widget.CoordinatorLayout>
```



Overview – *content_home* *

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="6dp"
    android:paddingLeft="6dp"
    android:paddingRight="6dp"
    android:paddingTop="6dp"
    app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingVie..."
    tools:context=".activities.Home"
    tools:showIn="@layout/app_bar_home">

    <TextView...>

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent" android:id="@+id/homeFrame"
        android:layout_above="@+id/footerLinearLayout"
        android:layout_below="@+id/recentAddedBarTextView" />

    <LinearLayout...>

</RelativeLayout>
```



Overview – *Bind to the Drawer Layout etc.*

- Once you have the necessary layouts and menu in place, you then need to bind to the **Drawer** and **Navigation View** to allow you to handle the user navigation and switching content based on user selection.
- In your **`onCreate()`** you'll have something like the following

```
DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
        this, drawer, toolbar, "Open navigation drawer", "Close navigation drawer");
drawer.addDrawerListener(toggle);
toggle.syncState();

NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
navigationView.setNavigationItemSelectedListener(this);
```

We also setup GooglePhoto and Email for the Drawer here (Labs)

Overview – *Bind to the Drawer Layout etc.*

- You'll probably want to display some kind of initial landing page once the app starts so in our example, we load up the list of user coffees (maintained in our **`CoffeeFragment`**).
- Again, in your *onCreate()* you'll have something like the following

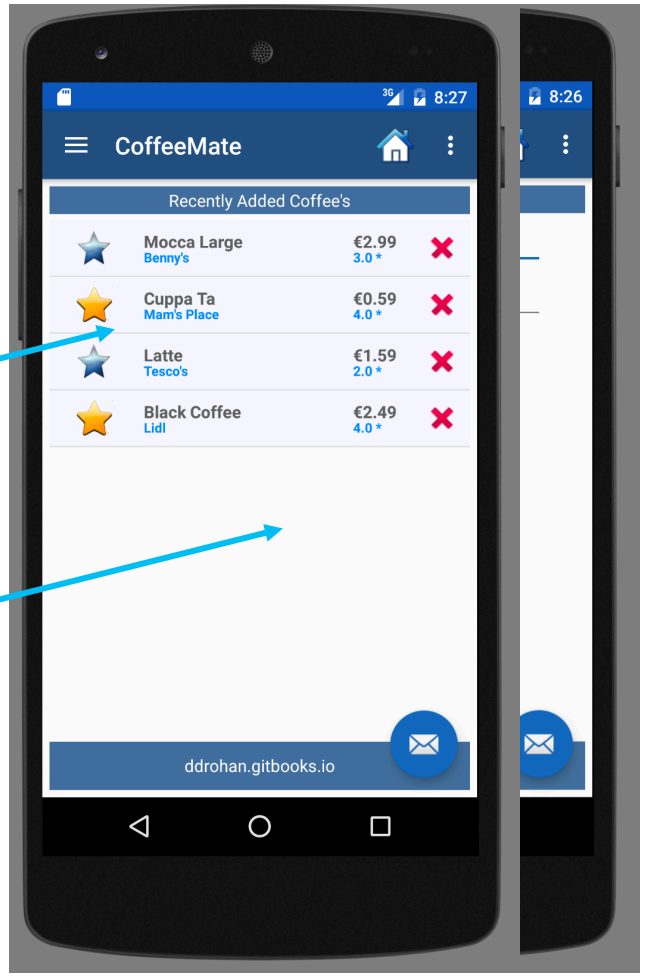
```
FragmentTransaction ft = getFragmentManager().beginTransaction();

CoffeeFragment fragment = CoffeeFragment.newInstance();
ft.replace(R.id.homeFrame, fragment);
ft.commit();
```

- This creates a new instance of a CoffeeFragment and replaces the fragment in our FrameLayout with this instance.

Overview – *Handle Drawer Click & Update Content* *

- To handle users menu selection we implement the following

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
    // http://stackoverflow.com/questions/32944798/switch-between-fragments
    int id = item.getItemId();
    Fragment fragment;
    FragmentTransaction ft = getFragmentManager().beginTransaction();

    if (id == R.id.nav_home) {
        fragment = CoffeeFragment.newInstance();
        ((CoffeeFragment)fragment).favourites = false;
        ft.replace(R.id.homeFrame, fragment);
        ft.addToBackStack(null);
        ft.commit();
    } else if (id == R.id.nav_add) {
        fragment = AddFragment.newInstance();
        ft.replace(R.id.homeFrame, fragment);
        ft.addToBackStack(null);
        ft.commit();
    }

    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}
```



Summary

- We looked at how to use **`Spinners`** and **`Filters`** to allow users to **Search** on our list of coffees
- We're now able to share data efficiently and easily between Activities using the **`Application`** object
- We reused **`Fragments`** in a multi-screen app to go '*Green*' – (Reduce,Reuse,Recycle)
- And we made use of a **`NavigationDrawer`** to implement more effective navigation within our app (**CoffeeMate 4.0+**)


Thanks!

Questions?

Features Not Used in the Case Study

- ❑ Spinners (setup via XML)
- ❑ Context menus (long Click)
- ❑ Notifications

Using Spinners

Approach : Choices in XML (NOT used in this Case Study)

- Idea
  - A combo box (drop down list of choices)
    - Similar purpose to a RadioGroup: to let the user choose among a fixed set of options
- Main Listener types
  - `AdapterView.OnItemSelectedListener`
  - `AdapterView.OnItemClickedListener`

    - The first is more general purpose, since it will be invoked on programmatic changes and keyboard events as well as clicks.

Approach (continued)

- Key XML attributes
  - android:id
    - You need a Java reference to assign an event handler
  - android:prompt
    - The text shown at the top of Spinner when user clicks to open it.
      - Since text is *not* shown when the Spinner is closed, the string used for the prompt is typically also displayed in a TextView above the Spinner.
  - android:entries
    - An XML entry defining an array of choices.
      Can be in strings.xml or a separate file (e.g., arrays.xml as in our case study)

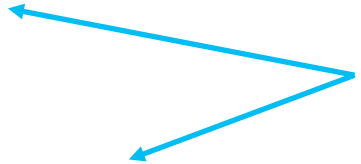
```
<string-array name="some_name">
    <item>choice 1</item>
    <item>choice 2</item>
    …
</string-array>
```

OnItemSelectedListener (interface)

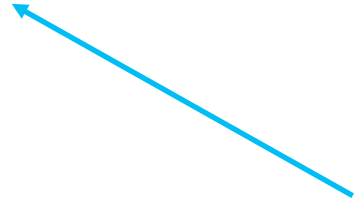
- onItemSelected
  - Invoked when an entry is selected. Invoked once when Spinner is first displayed, then again for each time the user selects something.
  - Arguments
    - AdapterView: the Spinner itself
    - View: the row of the Spinner that was selected
    - int: the index of the selection. Pass this to the Spinner's getItemAtPosition method to get the text of the selection.
    - long: The row id of the selected item
- onNothingSelected
  - Invoked when there is now nothing displayed. This cannot happen due to normal user interaction, but only when you programmatically remove an entry.

XML: Sample Layout File Entry

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/spinner1_prompt"/>
<Spinner
    android:id="@+id/spinner1"
    android:prompt="@string/spinner1_prompt"
    android:entries="@array/spinner1_entries"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```



Same text used twice, since the text is hidden when the Spinner is closed.



An array of entries. If you have lots of arrays, you typically put them in arrays.xml. However, if there's just the one set of choices, it makes more sense to keep the array of entries in strings.xml with the spinner prompt and the spinner message template.

XML: Sample Strings File Entries

```
<string name="spinner1_prompt">
   Current Android Vendors (Choices from XML)
</string>
<string-array name="spinner1_entries">
    <item>Acer</item>
    <item>Dell</item>
    <item>HTC</item>
    <item>Huawei</item>
    <item>Kyocera</item>
    <item>LG</item>
    <item>Motorola</item>
    <item>Nexus</item>
    <item>Samsung</item>
    <item>Sony Ericsson</item>
    <item>T-Mobile</item>
    <item>Neptune</item>
</string-array>
<string name="spinner_message_template">
    You selected \'%s\'.
</string>
```

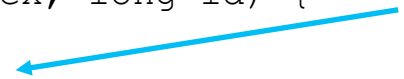
The event handler method will use String.format, this template, and the current selection to produce a message that will be shown in a Toast when a Spinner selection is made.

Java (Relevant Parts)

```
public class SpinnerActivity extends Activity {
    private String mItemSelectedMessageTemplate;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.spinners);
        mItemSelectedMessageTemplate =
                getString(R.string.spinner_message_template);
        Spinner spinner1 = (Spinner)findViewById(R.id.spinner1);
        spinner1.setOnItemSelectedListener(new SpinnerInfo());
       }

    private void showToast(String text) {
        Toast.makeText(this, text, Toast.LENGTH_LONG).show();
    }

    // Continued on next slide with the SpinnerInfo inner class
```

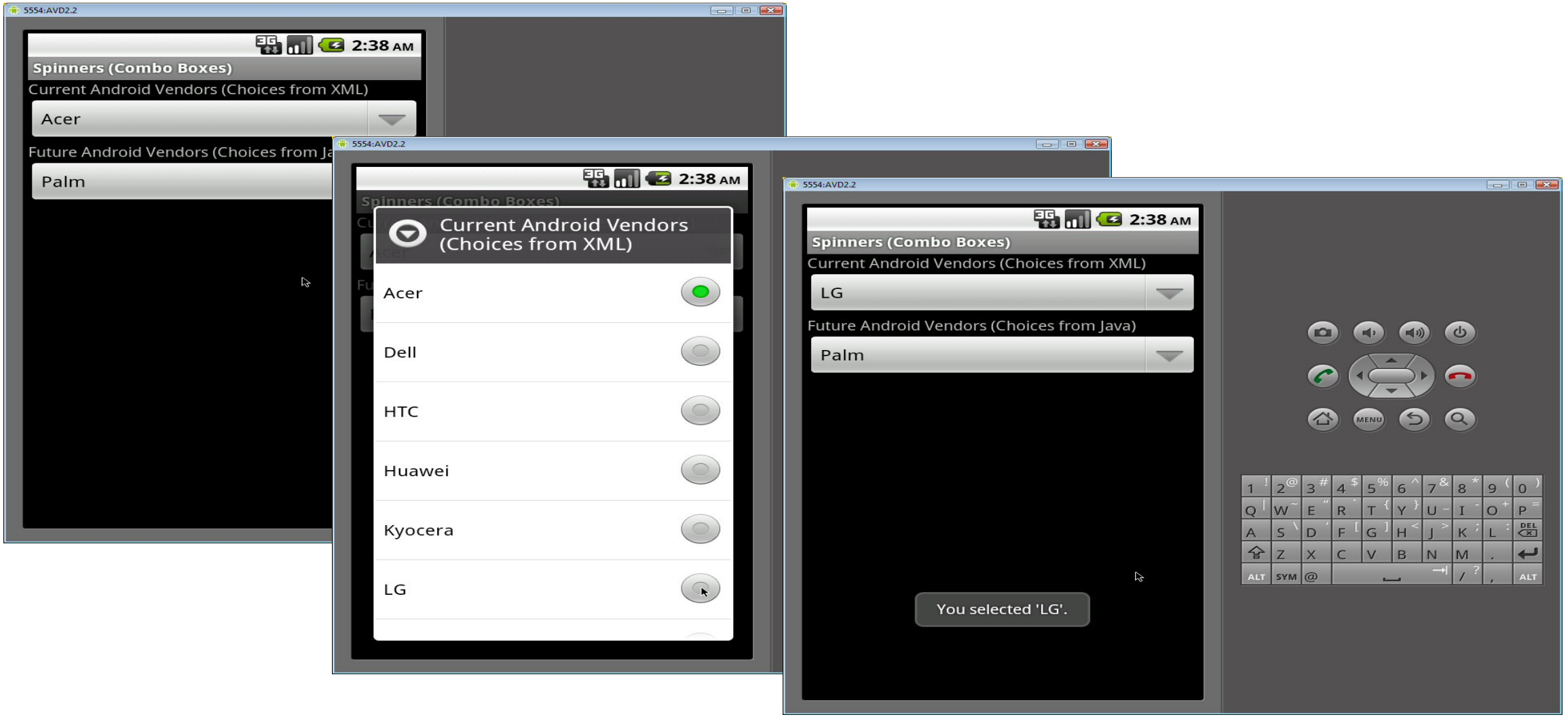
Java (Relevant Parts, Cont'd)

```
private class SpinnerInfo implements OnItemSelectedListener {
    private boolean isFirst = true;

   @Override
    public void onItemSelected(AdapterView<?> spinner, View selectedView,
                               int selectedIndex, long id) {
        if (isFirst) {
            isFirst = false;
        } else {
            String selection =
                    spinner.getItemAtPosition(selectedIndex).toString();
            String message =
                    String.format(mItemSelectedMessageTemplate, selection);
            showToast(message);
        }
    }

    @Override
    public void onNothingSelected(AdapterView<?> spinner) {
        // Won't be invoked unless you programmatically remove entries
    }
}
```



Don't want the Toast when the screen is first displayed, so ignore the first call to onItemSelected. Other calls are due to user interaction.

Results (Emulator)



Adding a Context Menu

Step 1 : Register View for a context menu

- By calling `registerForContextMenu()` and passing it a View (a TextView in this example) you assign it a context menu.
- When this View (TextView) receives a long-press, it displays a context menu.

```
public class DemoActivity extends Activity {

 /** Called when the activity is first created.
 */
 @Override
 public void onCreate(Bundle
 savedInstanceState) {
 super.onCreate(savedInstanceState);
```

Define menu's appearance

- By overriding the activity's context menu create callback method, `onCreateContextMenu()`.

```
public void onCreateContextMenu(ContextMenu menu,
  View v,                       ContextMenuInfo menuInfo) {

    MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.mymenu, menu);

  menu.setHeaderTitle("Please Choose an Option");
      menu.setHeaderIcon(R.drawable.myimage);
```

Same menu options but doesn't have to be

Set the context menus prompt and icon – both optional

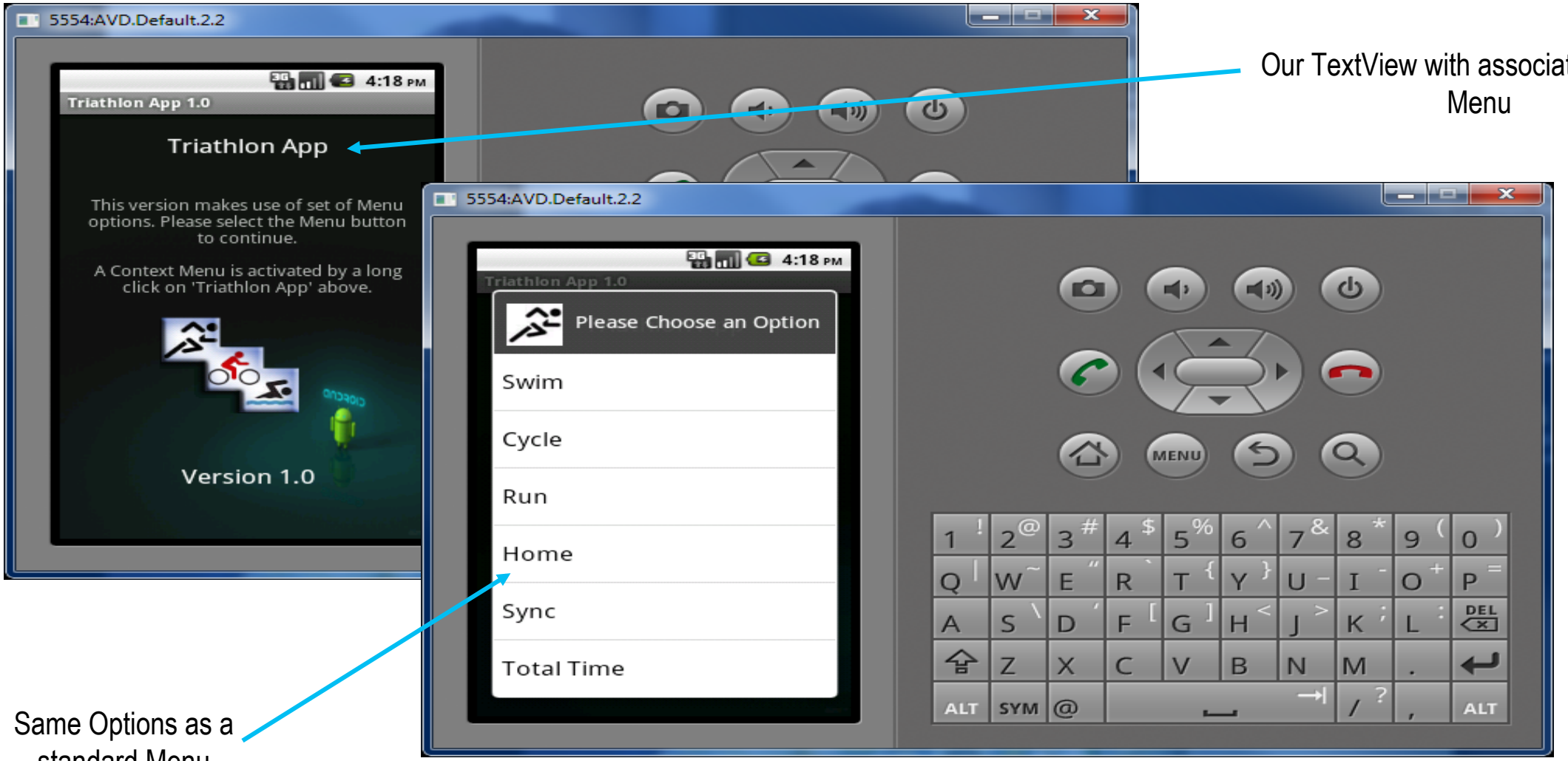
Define menu's behavior

- By overriding your activity's menu selection callback method for context menu , `onContextItemSelected()`.

```
public boolean
 onContextItemSelected(MenuItem item) {
Log.v("Context Menu","Item Selected : " +
        item.getTitle());
    return false;
}
```

We're just printing a message to the LogCat window, as we're using the same menu options, but you can put whatever you need here.

Results on Emulator – A Sample App



Our TextView with associated Context Menu

Same Options as a standard Menu

Status Bar Notifications

Status Bar Notifications (2)

- A status bar notification should be used for any case in which a background service needs to alert the user about an event that requires a response.
- A background service ***should never*** launch an activity on its own, in order to receive user interaction. The service should instead create a status bar notification that will launch the activity when selected by the user.

Status Bar Notifications (3)

To create a status bar notification:

1. Get a reference to the NotificationManager:

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mManager = (NotificationManager) getSystemService(ns);
```

2. Instantiate the Notification:

```
int icon = R.drawable.notification_icon;
CharSequence tickerText = "Hello";
long when = System.currentTimeMillis();

Notification notification = new Notification(icon, tickerText, when);
```

Status Bar Notifications (4)

3. Define the notification's message and PendingIntent:

A description of an Intent and target acti
to perform with it

```
Context context = getApplicationContext();
CharSequence contentTitle = "My notification";
CharSequence contentText = "Hello World!";
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
notificationIntent, 0);

notification.setLatestEventInfo(context, contentTitle, contentText,
contentIntent);
```

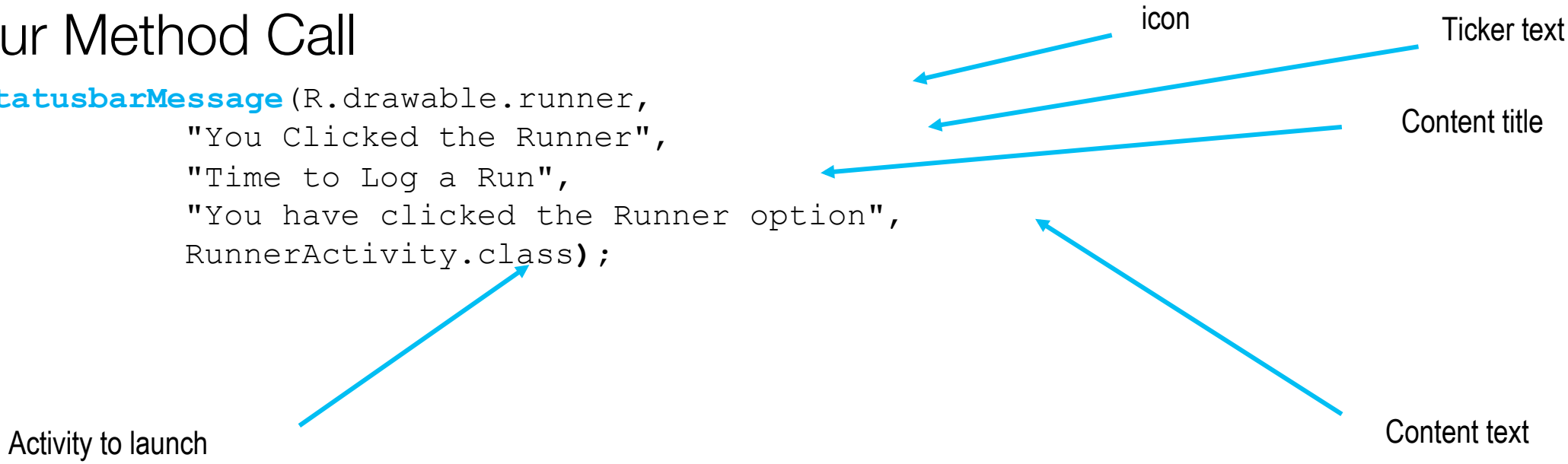
4. Pass the Notification to the NotificationManager:

```
private static final int HELLO_ID = 1;

mManager.notify(HELLO_ID, notification);
```

Status Bar Notifications (5)

Our Method Call

```
postStatusbarMessage(R.drawable.runner,
                "You Clicked the Runner",
                "Time to Log a Run",
                "You have clicked the Runner option",
                RunnerActivity.class);
```



Status Bar Notifications (6)

```
private void postStatusbarMessage(int icon, String tickerText,
String contentTitle, String contentText, Class<?> activityClass)
{
Intent notificationIntent = new Intent(this, activityClass);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
Context context = getApplicationContext();
long when = System.currentTimeMillis();

Notification notification = new Notification(icon,tickerText,when+5000);
notification.setLatestEventInfo(context, contentTitle, contentText, contentIntent);

String ns = Context.NOTIFICATION_SERVICE;

NotificationManager mManage = (NotificationManager)getSystemService(ns);

notification.flags |= Notification.FLAG_AUTO_CANCEL;

mManage.notify(ID, notification);
ID++;
}
```

Results on Emulator

Status Bar Notifications

Posted Notifications added when user selected a particular menu option (run or swim)