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

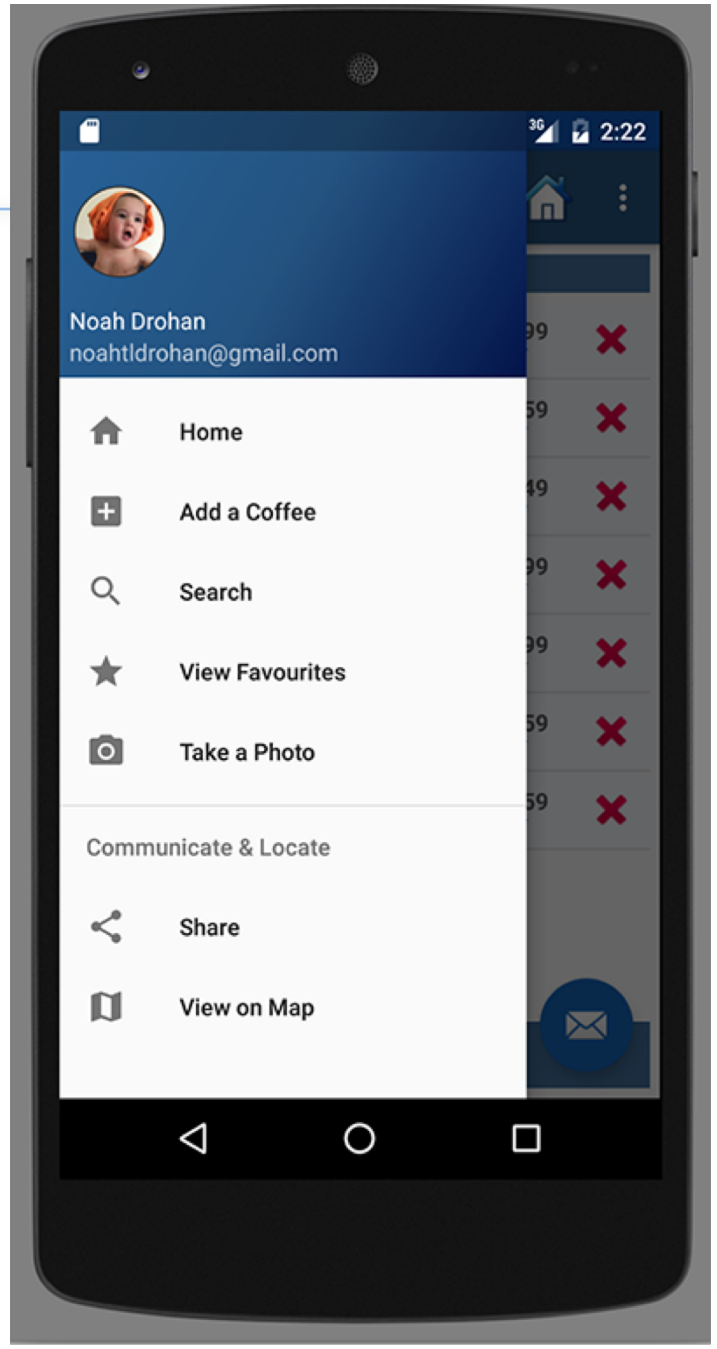
User Interface Design & Development – Part 2

Goals of this Section

- Be able to create and use some more different widgets (views) and features such as **`Spinners`**, **`SearchViews`** and **`Filters`**
- Share data between Activities using the **`Application`** object
- Understand how to develop and reuse **`Fragments`** in a multi-screen app

Case Study

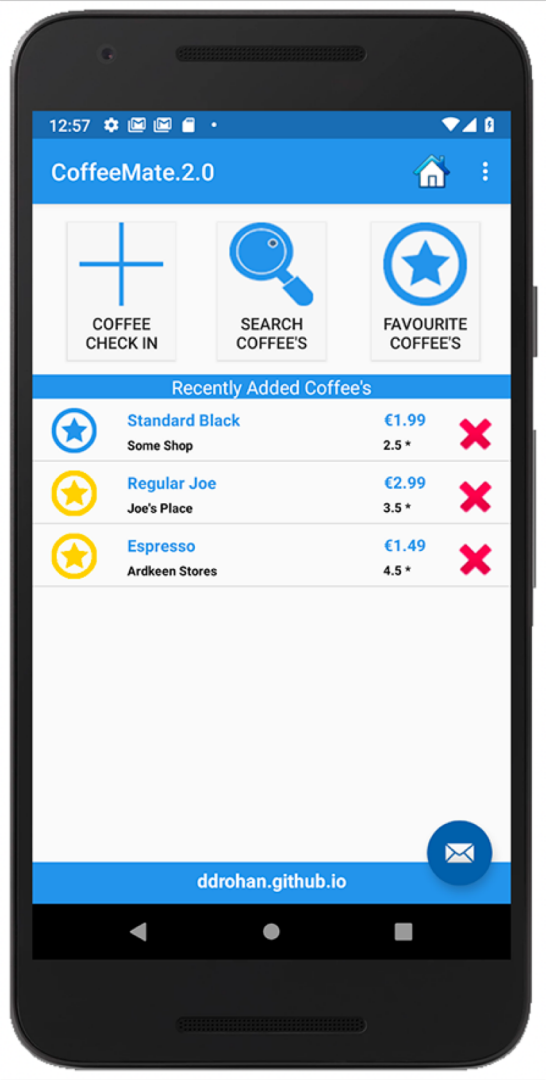
- *CoffeeMate* – an Android App to keep track of
  your Coffees, their details, and which ones you like the best (your favourites)
- App Features with Google+ Sign-In
  - List all your Coffees
  - View specific Coffee details
  - Filter Coffees by Name and Type
  - Delete a Coffee
  - List all your Favourite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

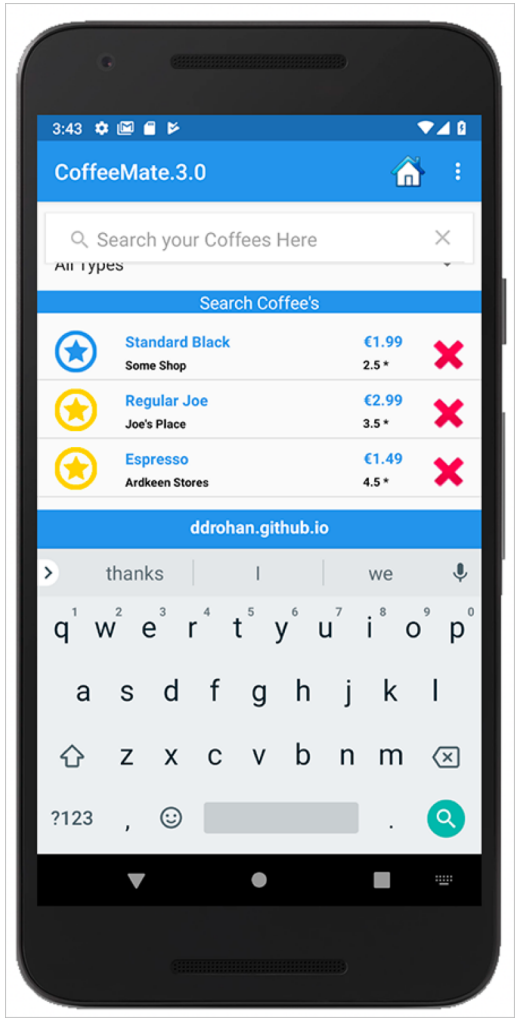


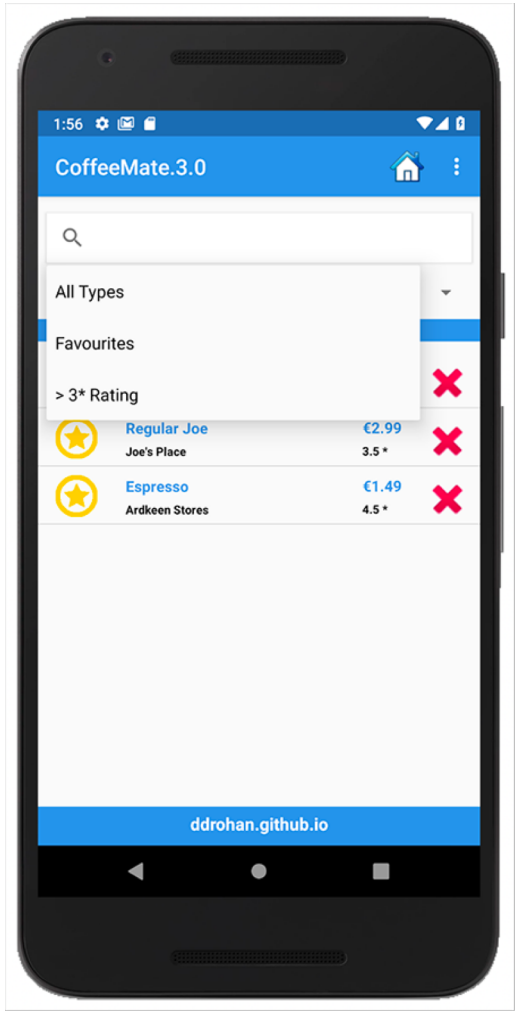
CoffeeMate 3.0

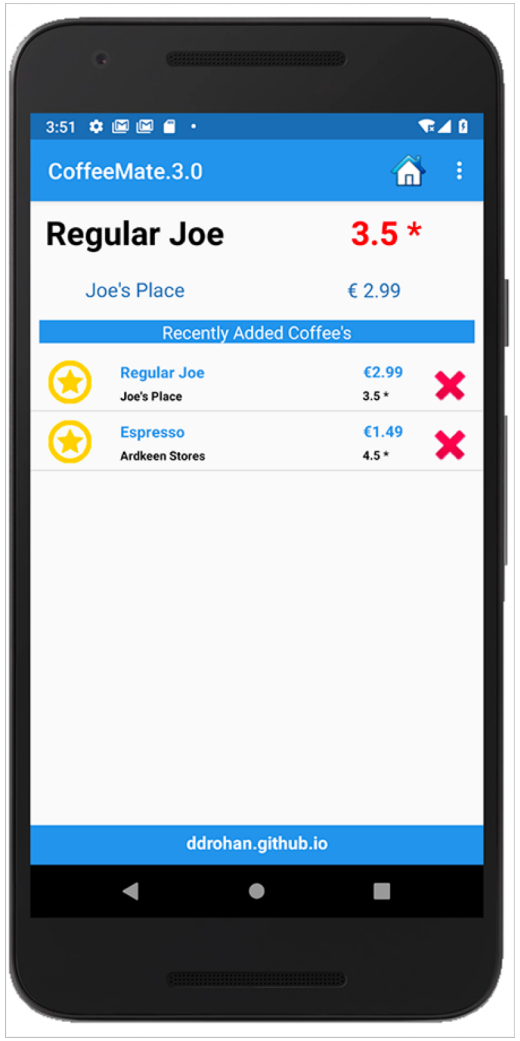
Using Spinners and Filters

CoffeeMate 3.0



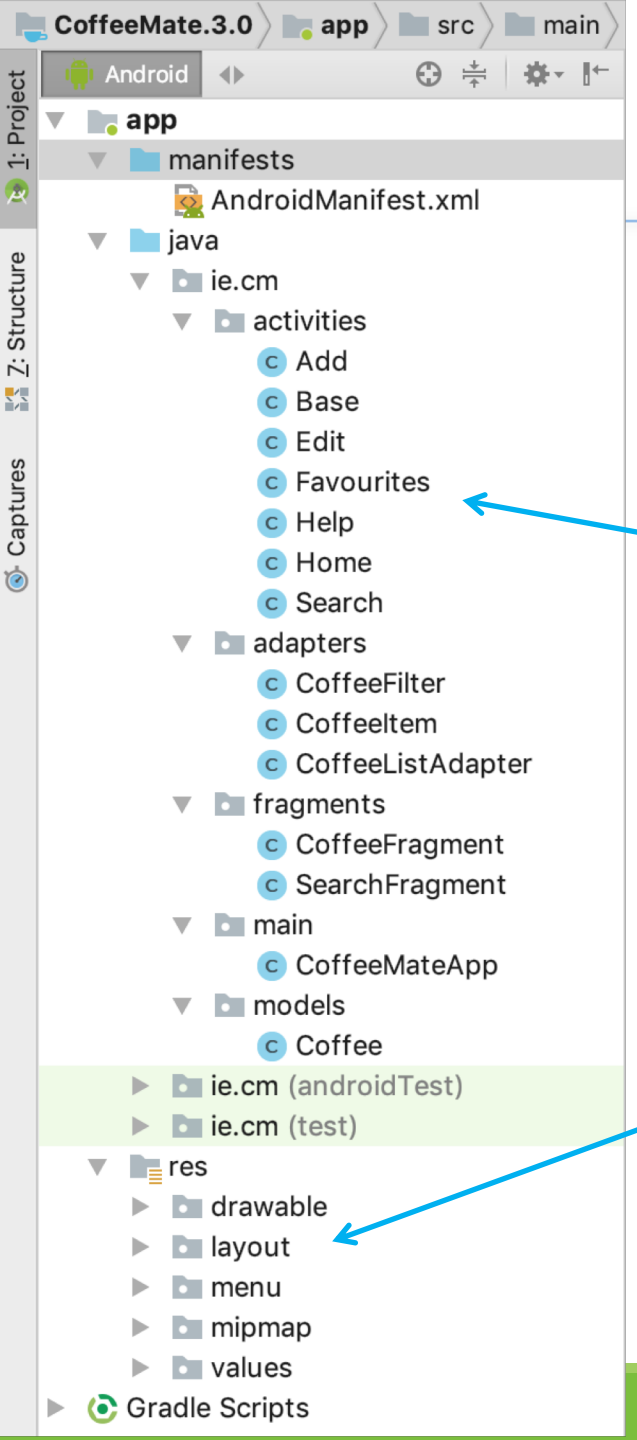






Still no Persistence in this Version

CoffeeMate 3.0



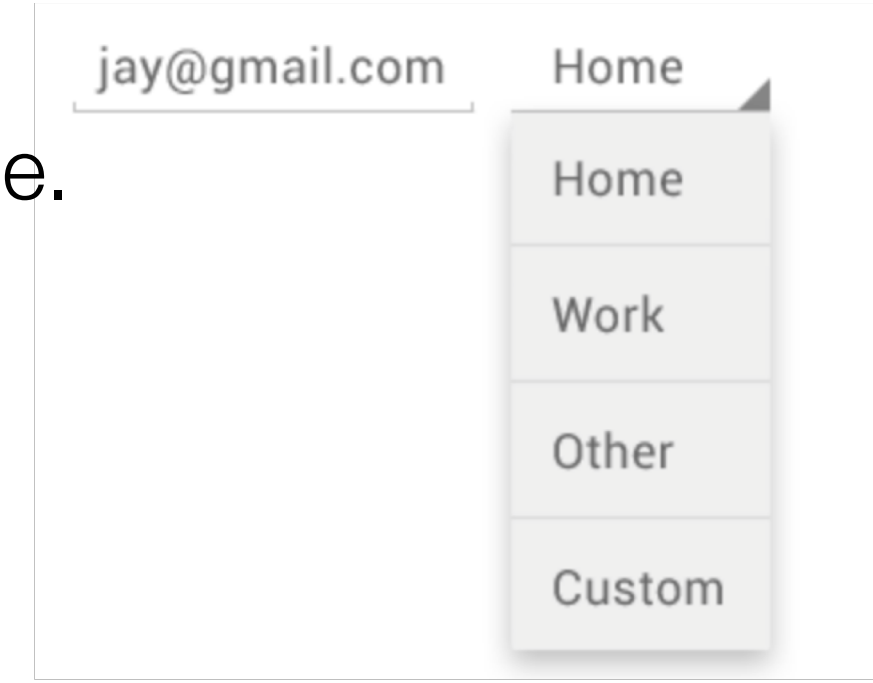
- 4 new java source files
- 2 new xml layouts

CoffeeMate 3.0

Using Spinners
&
SearchViews

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

```xml
<string-array name="coffeeTypes">
    <item>All Types</item>
    <item>Favourites</item>
    <item>\> 3\* Rating</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 = activity.findViewById(R.id.searchSpinner);
spinner.setAdapter(spinnerAdapter);
spinner.setOnItemSelectedListener(this);
```

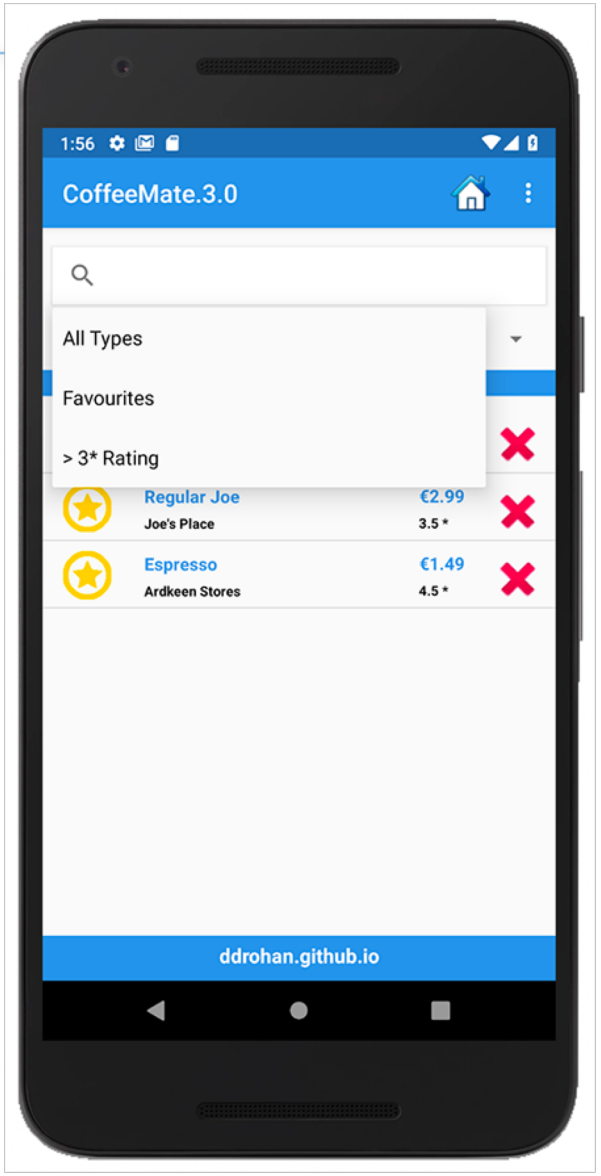
Populate the Spinner with User Choices

```
<string-array name="coffeeTypes">
    <item>All Types</item>
    <item>Favourites</item>
    <item>\> 3\* Rating</item>
</string-array>
```

This is the data we use to populate our spinner widget

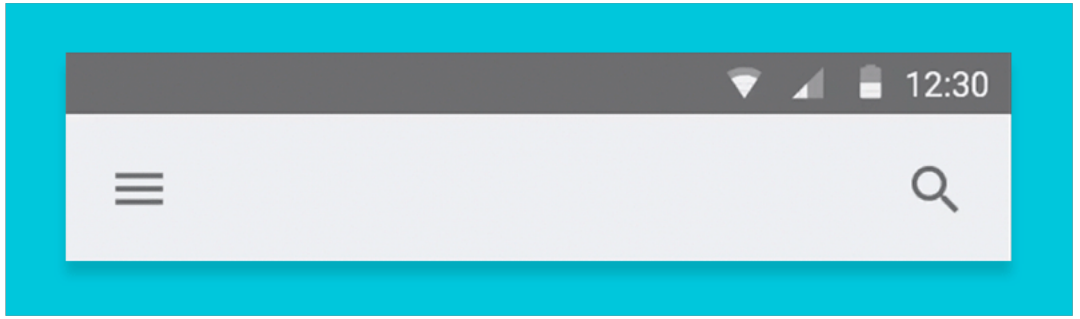
Key classes

- Spinner
- SpinnerAdapter
- AdapterView.OnItemSelectedListener



Overview – SearchViews *

- Since Android 3.0, the SearchView widget is the preferred way to provide a search in your app (Layout or App Bar)
- In its default state, a SearchView shows only its icon
- Depending on how you've configured it, touching the search icon will expand the widget to allow the user to enter search criteria, or suggest search items to select from – like so…



Overview – SearchViews

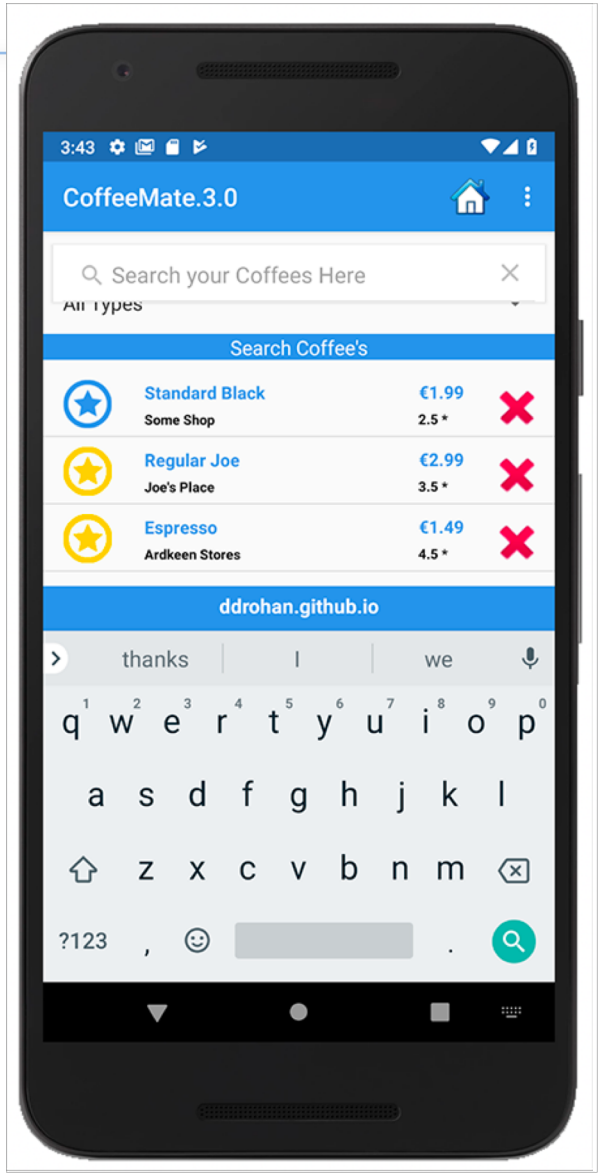
- You can add a SearchView to your Layout or App Bar with the **`SearchView`** object. You should usually do so in your XML layout with a <SearchView> element.

```
<SearchView
    android:id="@+id/searchView"
    android:layout_width="0dp"
    android:layout_height="46dp"
    android:layout_marginBottom="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
```

- There are many different ways for searching such as voice search, suggestions etc. Here we use the **`OnQueryTextListener`** and **`Filterable`** interfaces.

Overview – SearchViews

Nested classes	
interface	SearchView.OnCloseListener
interface	SearchView.OnQueryTextListener Callbacks for changes to the query text.
interface	SearchView.OnSuggestionListener Callback interface for selection events on suggestions.



Set up the SearchView Widget

- Bind to a **`SearchView`** widget and set its Listener

```
searchView = activity.findViewById(R.id.searchView);
searchView.setQueryHint("Search your Coffees Here");

searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {

    @Override
    public boolean onQueryTextSubmit(String query) {
        coffeeFilter.filter(query);
        return false;
    }

    @Override
    public boolean onQueryTextChange(String newText) {
        coffeeFilter.filter(newText);
        return false;
    }
});
```

CoffeeMate 3.0

Code Highlights (1)

Search *

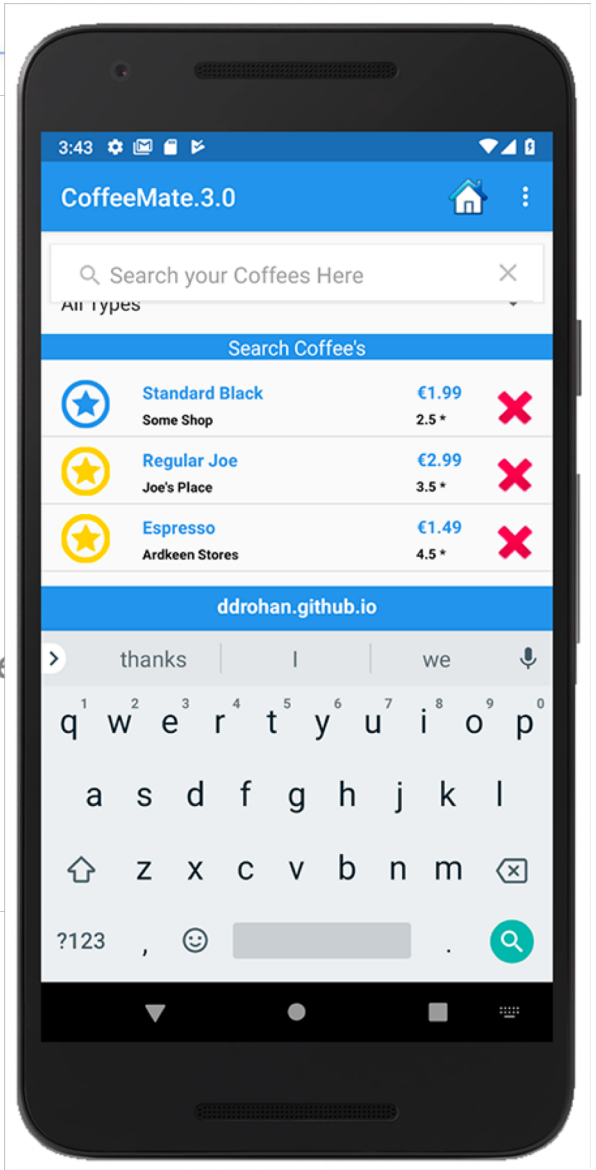
VERY similar to our **Home** Activity

```
public class Search extends Base  {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.search);
    }

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

        coffeeFragment = SearchFragment.newInstance(); //get a new Fragme
        getFragmentManager().beginTransaction()
                .replace(R.id.fragment_container, coffeeFragment)
                .commit(); // add it to the current activity
    }
}
```



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

Favourites *

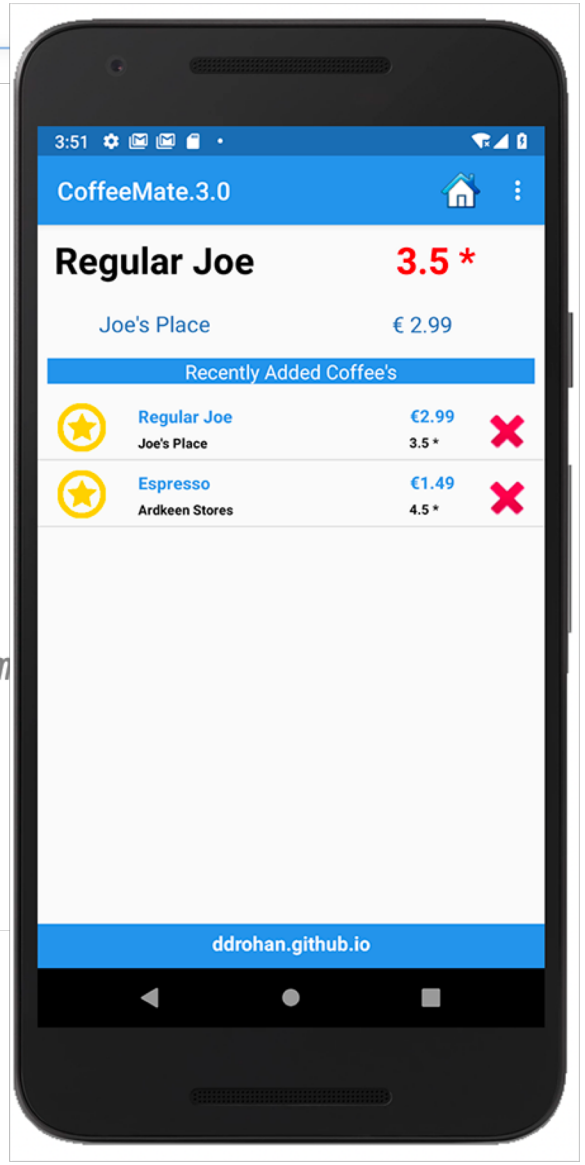
EXACTLY similar to our **Home** Activity

```
public class Favourites extends Base {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.favourites);
    }

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

        coffeeFragment = CoffeeFragment.newInstance(); //get a new Fragm
        getFragmentManager().beginTransaction()
                .replace(R.id.fragment_container, coffeeFragment)
                .commit(); // add it to 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 or even a SearchView to your layout and attach a Listener 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
            View.OnClickListener, AbsListView.MultiChoiceModeListener
{
    public          Base                activity;
    public static   CoffeeListAdapter   listAdapter;
    public          ListView            listView;
    public          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, activity.app.coffeeList);
    coffeeFilter = new CoffeeFilter(activity.app.coffeeList,"all",listAdapter);

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

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

CoffeeFragment – Filtering after Multiple Deletes *

```
public class CoffeeFragment  extends ListFragment implements
            View.OnClickListener, AbsListView.MultiChoiceModeListener
{
    public          Base                activity;
    public static   CoffeeListAdapter   listAdapter;
    public          ListView            listView;
    public          CoffeeFilter        coffeeFilter;
```

We need to modify the 'Favourites' listAdapter after deleting multiple coffees to get our remaining 'favourites'.

```
public void deleteCoffees(ActionMode actionMode)
{
    for (int i = listAdapter.getCount() -1 ; i >= 0; i--) {
        if (listView.isItemChecked(i)) {
            activity.app.coffeeList.remove(listAdapter.getItem(i));
            if (activity instanceof Favourites)
                listAdapter.coffeeList.remove(listAdapter.getItem(i));
        }
    }
    setRandomCoffee();
    listAdapter.notifyDataSetChanged(); // refresh adapter
    checkEmptyList();

    actionMode.finish();
}
```

SearchFragment – Filtering after Multiple Deletes *

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

    String selected;
    SearchView searchView;
```

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

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

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 = ((SearchView)activity
                .findViewById(R.id.searchView)).getQuery().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 SearchView 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

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

CoffeeFilter (2) *

```
@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 Spinner Selection (*no text entered*) – Triggered by

```
searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {

    @Override
    public boolean onQueryTextSubmit(String query) {
        coffeeFilter.filter(query);
        return false;
    }

    @Override
    public boolean onQueryTextChange(String newText) {
        coffeeFilter.filter(newText);
        return false;
    }
});
```

Filtering on *Text* – Triggered by

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

    List<Coffee> newCoffees;
    String coffeeName;

    if (prefix == null || prefix.length() == 0) {
        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();
            newCoffees = new ArrayList<>();

            for (Coffee c : originalCoffeeList) {
                coffeeName = c.coffeeName.toLowerCase();
                if (coffeeName.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;
}
```

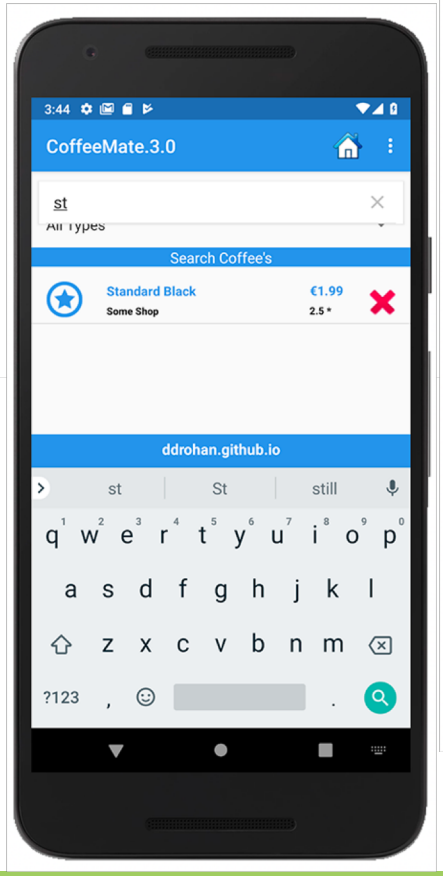
CoffeeFilter (3) *

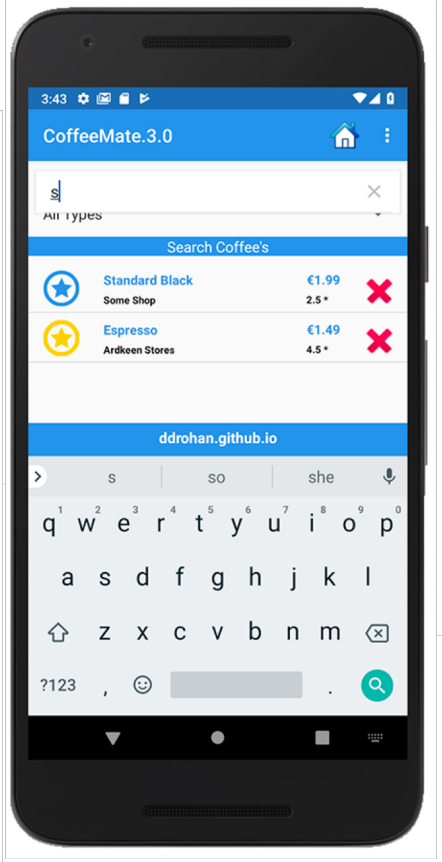
Invoked in the UI thread to publish the filtering results on the main UI thread (usually the user interface)

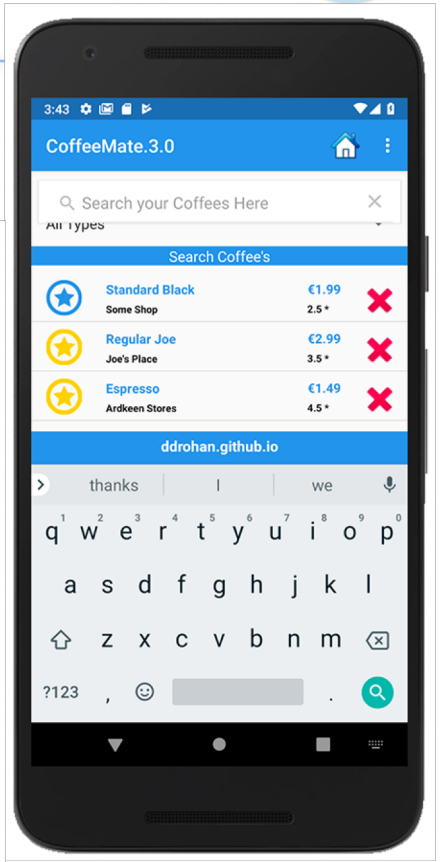
```
@Override
protected void publishResults(CharSequence prefix, FilterResults results) {

    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.

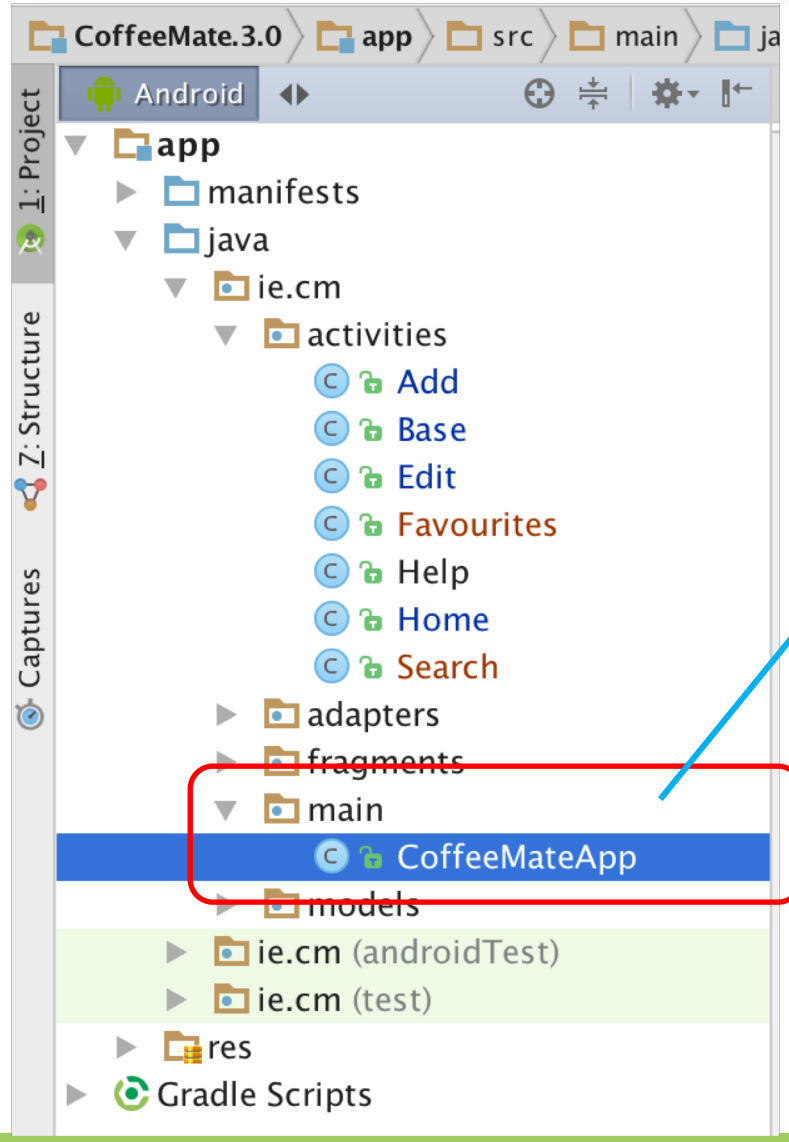
Refactor existing Activities/Classes

- 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 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<>();

    @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);
    startActivity(new Intent(this, Home.class));
```

Adding a Coffee to our coffeeList via the Application Object

Summary

- We looked at how to use **`Spinners`** , **`SearchViews`** 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)

Questions?


Thanks!

Features Not Used in the Case Study

- ❑ Notifications

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:

```
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);
```

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

4. Pass the Notification to the NotificationManager:

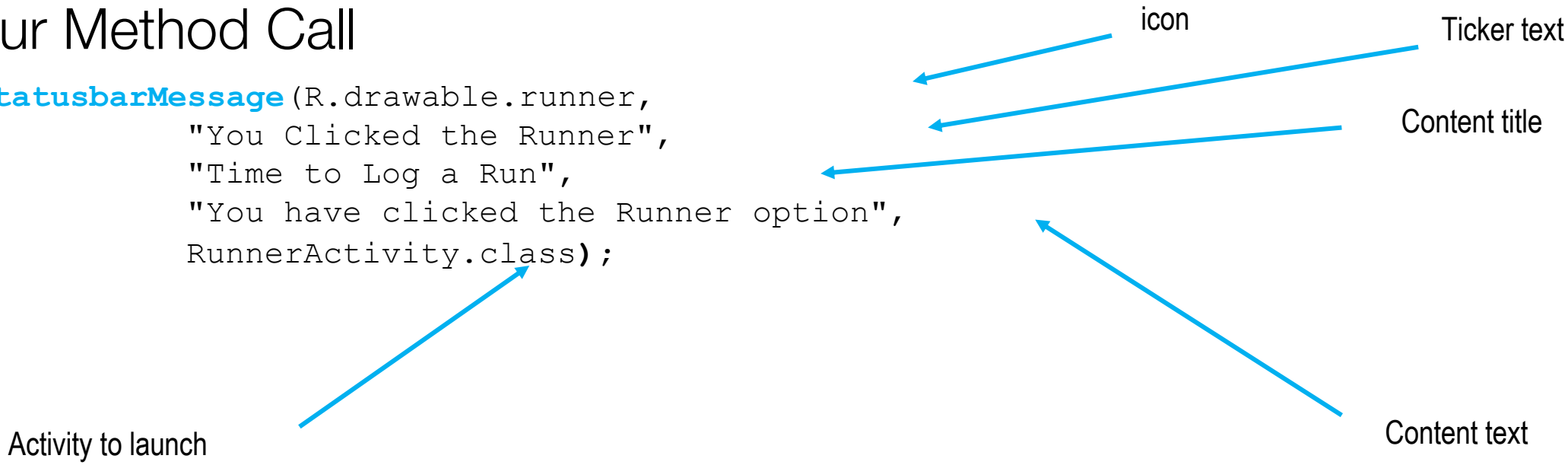
```
private static final int HELLO_ID = 1;

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)