Mobile and Ubiquitous Computing: Android Programming (part 3)

Master studies, Winter 2015/2016

Dr Veljko Pejović
Veljko.Pejovic@fri.uni-lj.si

Based on “Programming Handheld Systems”,
Adam Porter, University of Maryland
Examples from: “Smartphone Programming”
Andrew Campbell, Dartmouth College

University *of Ljubljana*
Faculty *of Computer and Information Science*

Android Security

- Applications are sandboxed
  - Each app has its user ID and group ID that is unique for the device
  - Allocated resources for the user ID
- Only the most basic functionalities are available to an application, other functionalities have to be explicitly asked for

Permissions

- An app requests permissions to access:
  - User data (e.g. contacts)
  - Some cost-sensitive APIs (e.g. send SMS)
  - Some system resources (e.g. camera)
- Permissions are declared in AndroidManifest.xml
  - Predefined Strings from Manifest.permission
  - <user-permission> in Manifest indicates a request:

```
<uses-permission android:name=
“android.permission.ACCESS_FINE_LOCATION”/>
```

Permissions Types

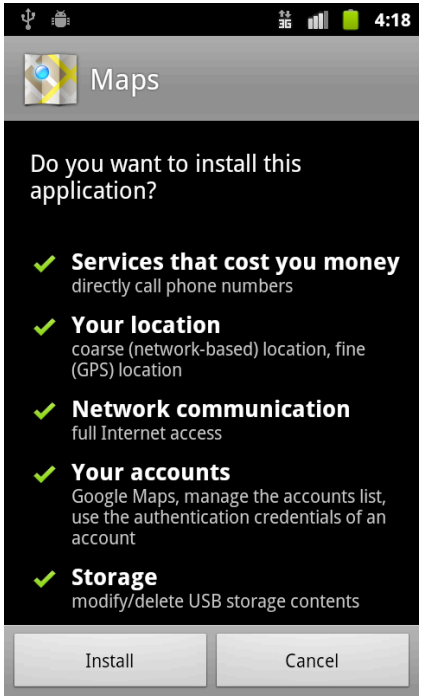
- Normal permissions
  - FLASHLIGHT, VIBRATE, BLUETOOTH, etc.

  http://developer.android.com/guide/topics/security/normal-permissions.html
- Dangerous permissions
  - READ_CONTACTS, SEND_SMS, ACCESS_COARSE_LOCATION, WRITE_EXTERNAL_CONTACTS, etc.
- The OS will explicitly ask the user to grant access for dangerous permissions

Handling Permissions

- Prior to Marshmallow (API 23):
  - Users must accept permissions before the app starts
- Marshmallow (API 23):
  - Permissions can also be granted at the runtime, when the functionality is about to be used
  - Ask straight away, educate up front, ask in context, educate in context

See a short explanation by Google devs:
www.youtube.com/watch?v=iZqDdvhTZj0



Custom Permissions

- Define your own permission if the app performs a privileged/dangerous function, and you don't want just any app to be able to launch yours
- In AndroidManifest.XML under <permission>
- Example:

```
<permission android:name="com.testpackage.mypermission"
            android:label="my_permission"
            android:protectionLevel="dangerous"/>
```

- Any app that wants to launch yours must request "com.testpackage.mypermission"

Component Permissions

- Permissions can be defined for individual components restricting which other component can call them
  - Activities
    - Which components can call startActivity()
  - Services
    - Which components can start or bind to a Service
  - BroadcastReceivers
    - Which components can receive and send broadcasts
  - ContentProviders
    - Which components can read and write content provider data

Permissions Design

- Do not request permissions unless you really need them
  - E.g. do you need to write data to an external storage (file) or can you keep the information in SharedPrefs?
- Show immediate benefit of granting a permission
  - E.g. your ToDo list app requires location info – show the user how she can make location-based reminders
- Use Intents to call other apps in case you don't need to handle the functionality within your app:
  - E.g. call a camera app, rather than requesting the camera permission for your app

Google Map in Android

- Provided by Google Play Services
  - Add to Android Studio via SDK Manager
- Needs maps API key
  - Obtain your apps short signature (SHA-1)
  - Create an API project in the Google API console
  - Generate the maps API key
  - Put it in your app's AndroidManifest.xml

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value=YOUR_KEY/>
```

Google Map in Android

- Android Studio automates this via New->Google Maps Activity
- Google maps require permissions

```
<uses-permission android:name=
    "android.permission.INTERNET" />
<uses-permission android:name=
    "android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name=
    "android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name=
    "com.google.android.providers.gsf.permission.READ_GSERVICES" />
```

Google Map in Android

- Map is basically a Fragment
  - com.google.android.gms.maps.SupportMapFragment
  - Note: you need Internet connectivity to show it
- Operations:
  - Set map type (satellite, normal, hybrid, terrain)
  - Capture onClick events
  - Add markers, capture onMarkerClick events

Google Map in Android

- You can do much more:
  - Draw on the map
    - Different layers
  - Get address
  - Show user's location
  - Get directions
  - Show 3D view (in selected areas)

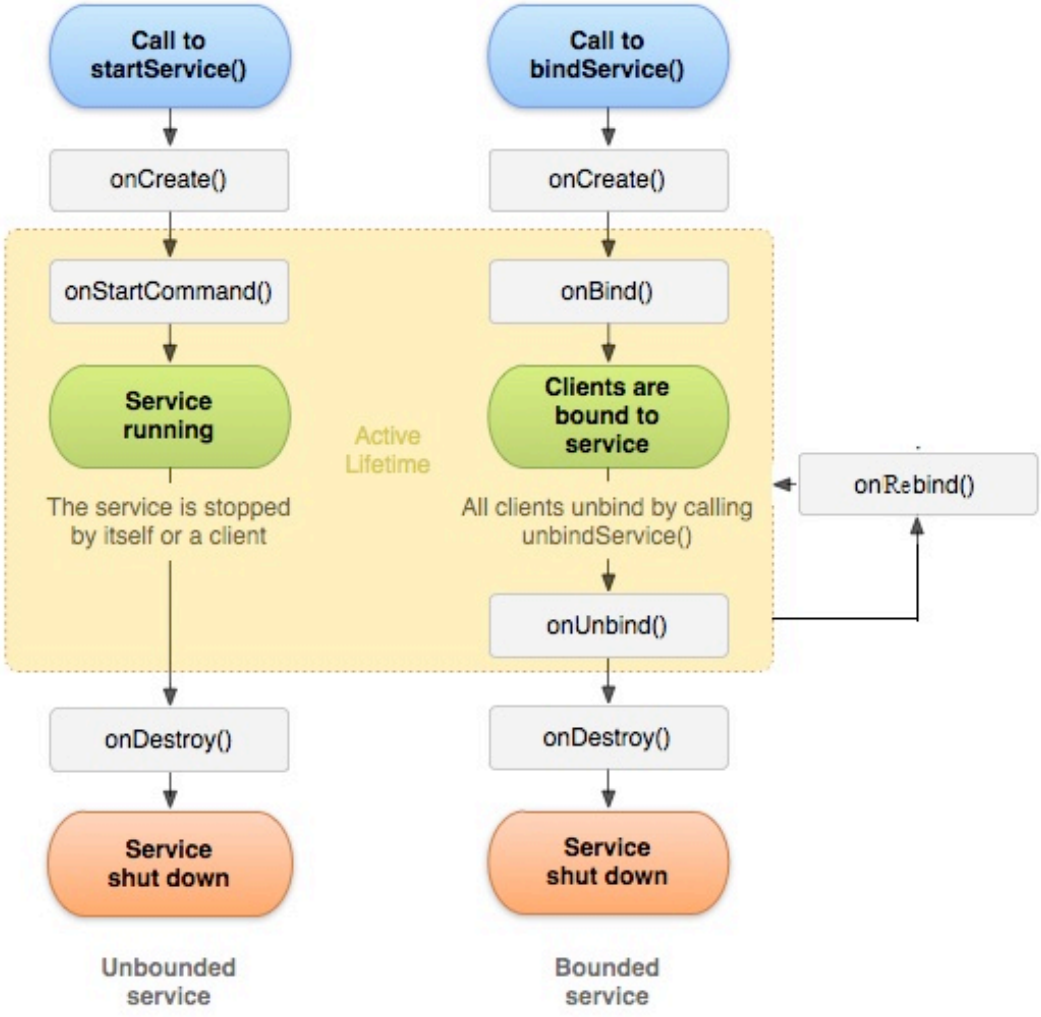
Try at home:
http://android-er.blogspot.si/2013/01/google-maps-android-api-v2-example_5213.html

Services

- Activities run on the main thread – UI thread
  - Processing-heavy functions on the main thread impact the responsiveness
- Services run on a separate background thread
  - Outside UI, for long-running operations
- Services can be created:
  - Explicitly using Context.startService()
  - Implicitly, if not already running, when a client requests connection to a Service via Context.bindService()

Services

- Multiple startService calls do not nest – you only have one service; however, onStartCommand() will be called repeatedly
- Service will be stopped only once with Context.stopService() or stopSelf()



Services – Bound

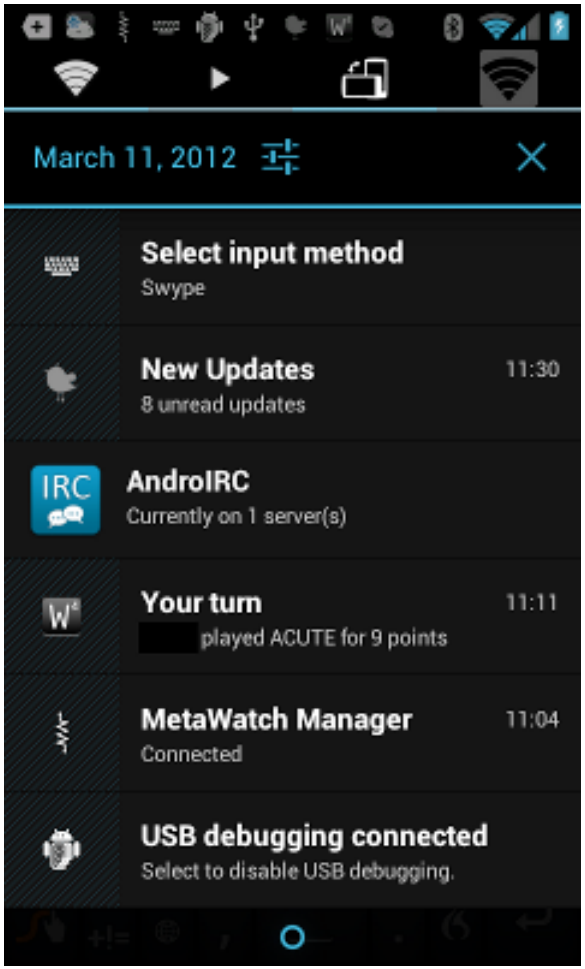
- Services started through binding do not call onStartCommand()
- Return IBinder object from onBind(Intent) so that connected clients can call the Service
- The service remains running as long as the connection is established

BroadcastReceiver

- Receive events announced by other components
- Events announced via Intents
  - Not the same Intent as the one starting an Activity: this one remains in the background
- Events can be announced within your app or publicly to every app on the phone
  - Announce via sendBroadcast()
- Events captured if the receiver is registered:
  - onReceiverRegistered() and then onReceive()

Notifications

- Allows the app to initiate contact with the user
- Shown in the Status Bar (Notification Bar)
  - Custom icons
  - Alerts: sound, vibration, LED flashing
  - Notification drawer
- NotificationManager

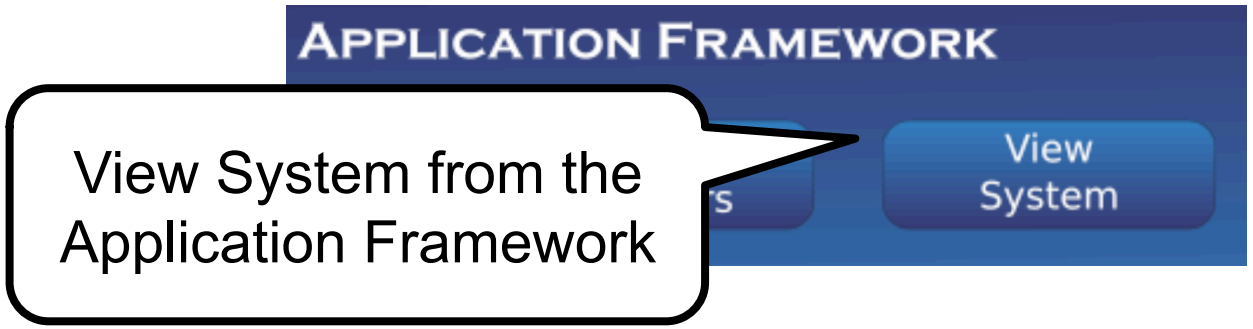


Service, Notification, BroadcastReceiver Example

University *of Ljubljana*
Faculty *of Computer and Information Science*

Android UI

- UI is usually provided via Activities and Fragments
  - setContentView(View v)



View

- View is the main class on which the View System operates
- Responsible for drawing itself and handling events
- From View we derive other UI objects: Button, EditText fields, MapView, RadioButton, WebView, LinearLayout, etc.
- View and its derived classes can be instantiated:
  - through the XML code in Resources
  - through Java source code

View – Example

- Button:

In an XML layout:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/reg_button_text"
    android:id="@+id/register_button"
    android:layout_gravity="center_horizontal”
/>
```

The benefit of using XML – separate functionality from presentation (think MVC)

View – Example

- Button:

In Activity through Java code :

```
final Button button = (Button)
findViewById(R.id.register_button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Do something, i.e. register a user
    }
});
```

Yet, accessing through Java brings a lot of flexibility

Other View Examples

- TextView
- EditText
- CheckBox
- SeekBar
- Switch
- CalendarView

Other View Examples

- AdapterViews
  - Separation between the data (model) and children views
  - Adapter manages the data and provides it to the views
  - Example: ListView
    - Displays a scrollable list of items, where the list is populated through an Adapter
    - ArrayAdapter is the most common, but you can implement your own Adapter as well
    - Convenience classes: ListActivity and ListFragment

View Properties

- View parameters
  - Text, padding, layout, colour, etc.
  - Layout: WRAP_CONTENT vs MATCH_PARENT

```
android:layout_width="wrap_content”
android:layout_height="wrap_content”
android:text="@string/reg_button_text”
android:id="@+id/register_button”
android:layout_gravity="center_horizontal”
```

Also accessible through Java: change visibility, modify text, get checked state, etc.

View Events

- Stem from user interaction and lifecycle changes
- Listeners – used for handling View events
  - OnClickListener.onClick() capture View clicked event
  - OnLongClickListener.onLongClick() capture View clicked and held for some time
  - OnTouchListener.onTouch() capture View touched
  - And quite a few more

Capturing gestures:
GestureDetector.OnGestureListener

View Events

- Example – Capture onClick event

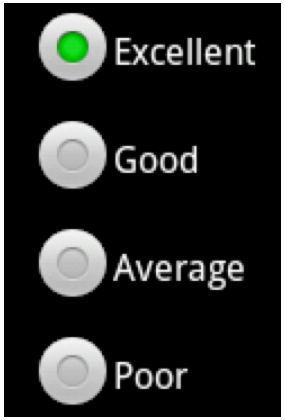
```
private OnClickListener mListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.button);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mListener);
    ...
}
```

Displaying Views

- Views are organised in a tree with the root View containing all the others
  - UI Hierarchy Viewer in Android Studio allows you to examine the hierarchy of your views
- Displaying Views includes
  - Measuring each of the elements (calling onMeasure())
  - Aligning the children (calling onLayout())
  - Rendering the view (onDraw())

Grouping Views

- ViewGroup
  - An invisible View that contains other Views
  - Used for grouping and organizing views
  - Example: RadioGroup, TimePicker, Spinner
- Layout
  - A ViewGroup that defines a structure for the Views it contains



Layouts

- LinearLayout
  - Child views arranged in a single horizontal or vertical row
- RelativeLayout
  - Child views are positioned relative to each other and to parent view
- TableLayout
  - Child views arranged into rows and columns
- But also other layouts: FrameLayout, GridLayout, TabLayout, etc.

Menus

- Formalised in Android to provide consistent user experience across applications
- Activities support menus:
  - Add items to a menu
  - Handle onClick events for the menu
- Menu types:
  - Options menu
  - Context menu
  - Popup menu

Options Menu

- Primary menu where actions such as “compose new email”, “settings” and similar are added
- Define in an XML file, put in res/menu

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

Options Menu

- Inflate in Activity- override onCreateOptionsMenu

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

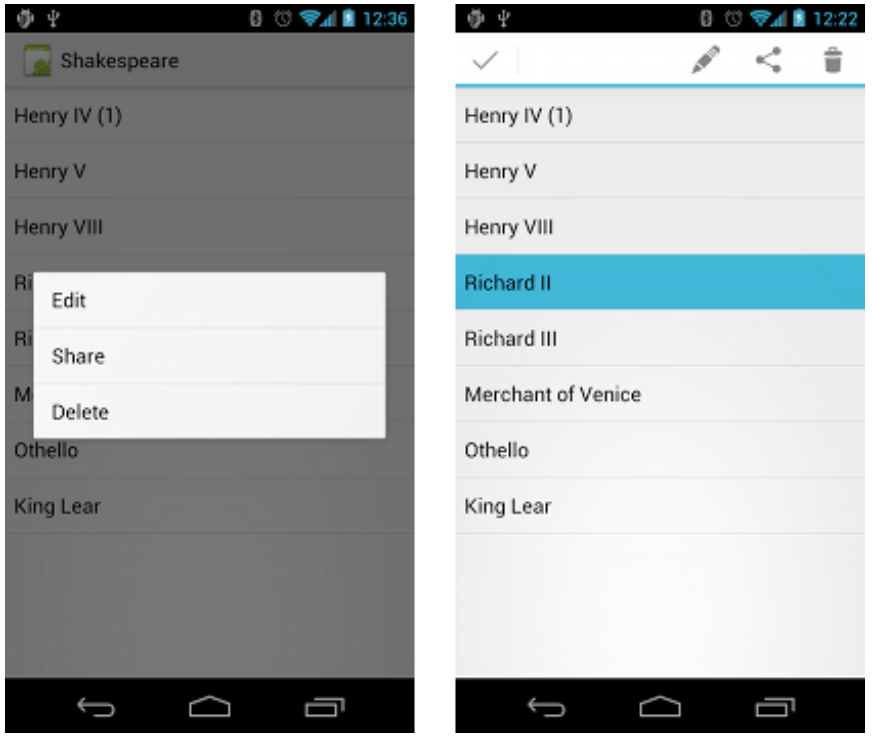
Note: Fragments have their own onCreateOptionsMenu

- Handle onClick events

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
```

Context Menu

- Options that affect the selected item in the context of the current UI frame
- Floating context menu
- Contextual action mode



Context Menu

- Define your menu in res/menu as an XML file
- Register a View for a context menu

```
Button btn = (Button) findViewById(R.id.btn);
registerForContextMenu(btn);
```

- Inflate context menu

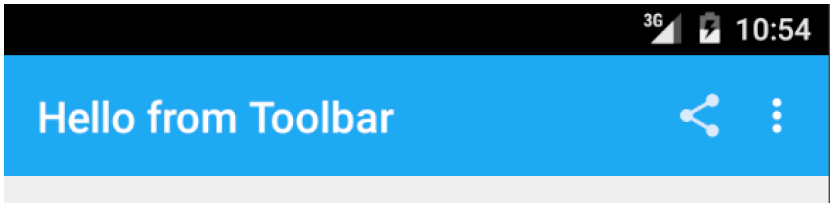
```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);}
```

- Capture on item selected events

```
public boolean onContextItemSelected(MenuItem item) …
```

Toolbar

- Allows quick access to frequently used operations
  - Title, logo, navigation, action menu



- Similar to the application bar in desktop apps
- Toolbar - Introduced in API 21 (Lollipop) to replace ActionBar introduced in API 11
  - Toolbar is more general than ActionBar (which is closely tied to an Activity), as it can be placed anywhere within the view hierarchy

Use Toolbar as ActionBar

```
dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  compile "com.android.support:appcompat-v7:21.0.+" }
```

Gradle

```
<android.support.v7.widget.Toolbar
  android:id="@+id/toolbar"
  android:layout_width="match_parent"
  android:layout_height="wrap_content">
</android.support.v7.widget.Toolbar>
```

Layout XML

```
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;

public class MyActivity extends AppCompatActivity{
   @Override protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_my);
       Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
       setSupportActionBar(toolbar);
   }
}
```

Activity Java