Android – An Overview

Mihai Fonoage

February 10, 2009

Outline

- Introduction
- Application Components and Lifecycle
- User Interfaces
- Binding applications and their components
- Data Storage
- Background Services
- Location-Based Services
- Accessing Android’s Sensors
- References

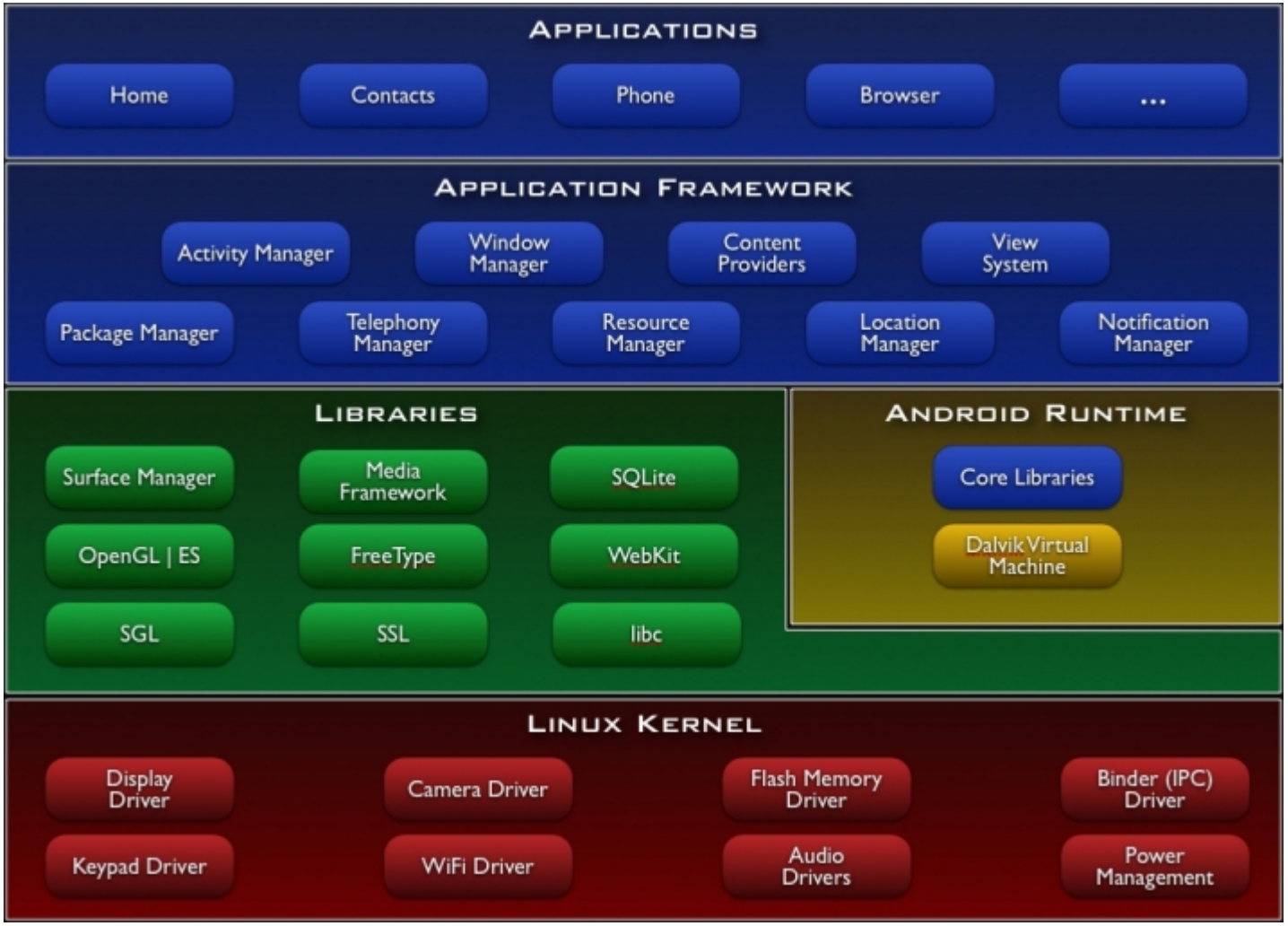
Outline

- Introduction
- Application Components and Lifecycle
- User Interfaces
- Binding applications and their components
- Data Storage
- Background Services
- Location-Based Services
- Accessing Android’s Sensors
- References

Introduction

- Android is a software stack for mobile devices that includes:
  - Operating System
    - Linux version 2.6
      - Services include hardware drivers; power, process and memory management; security and network.
  - Middleware
    - Libraries (i.e. SQLite, OpenGL, WebKit, etc)
    - Android Runtime (Dalvik Virtual Machine and core libraries)
    - Application Framework
      - Abstraction for hardware access; manages application resources and the UI; provides classes for developing applications for Android
  - Applications
    - Native apps: Contacts, Phone, Browser, etc.
    - Third-party apps: developer's applications.

Introduction (cont.)



Source: http://code.google.com/android/what-is-android.html

Introduction (cont.)

- What you need:
  - Operating System: Microsoft Windows (>= XP), Mac OS X >= 10.4.8, Linux
  - Android SDK
  - JDK >= 5
- Android Development with Eclipse:
  - Eclipse (+ Java Development Tools plug-in and Web Tools Platform) + Android Development Tools plug-in
- Installation notes: http://code.google.com/android/intro/installing.html.

Introduction (cont.)

- Design Considerations:
  - Low processing speed
    - Optimize code to run quick and efficiently
  - Limited storage and memory
    - Minimize size of applications; reuse and share data
  - Limited bandwidth and high latency
    - Design your application to be responsive to a slow (sometimes non-existent), intermittent network connection
  - Limited battery life
    - Avoid expensive operations
  - Low resolution, small screen size
    - “Compress” the data you want to display

Outline

- Introduction
- Application Components and Lifecycle
- User Interfaces
- Binding applications and their components
- Data Storage
- Background Services
- Location-Based Services
- Accessing Android’s Sensors
- References

Application Components and Lifecycle

- Components of your application:
  - **Activities**
    - Presentation layer for the application you are building
    - For each screen you have, their will be a matching Activity
    - An Activity uses Views to build the user interface
  - **Services**
    - Components that run in the background
    - Do not interact with the user
    - Can update your data sources and Activities, and trigger specific notifications

Android Application Overview (cont.)

- Components of your application:
  - **Content Providers**
    - Manage and share application databases
  - **Intents**
    - Specify what intentions you have in terms of a specific action being performed
  - **Broadcast Receivers**
    - Listen for broadcast Intents that match some defined filter criteria
    - Can automatically start your application as a response to an intent

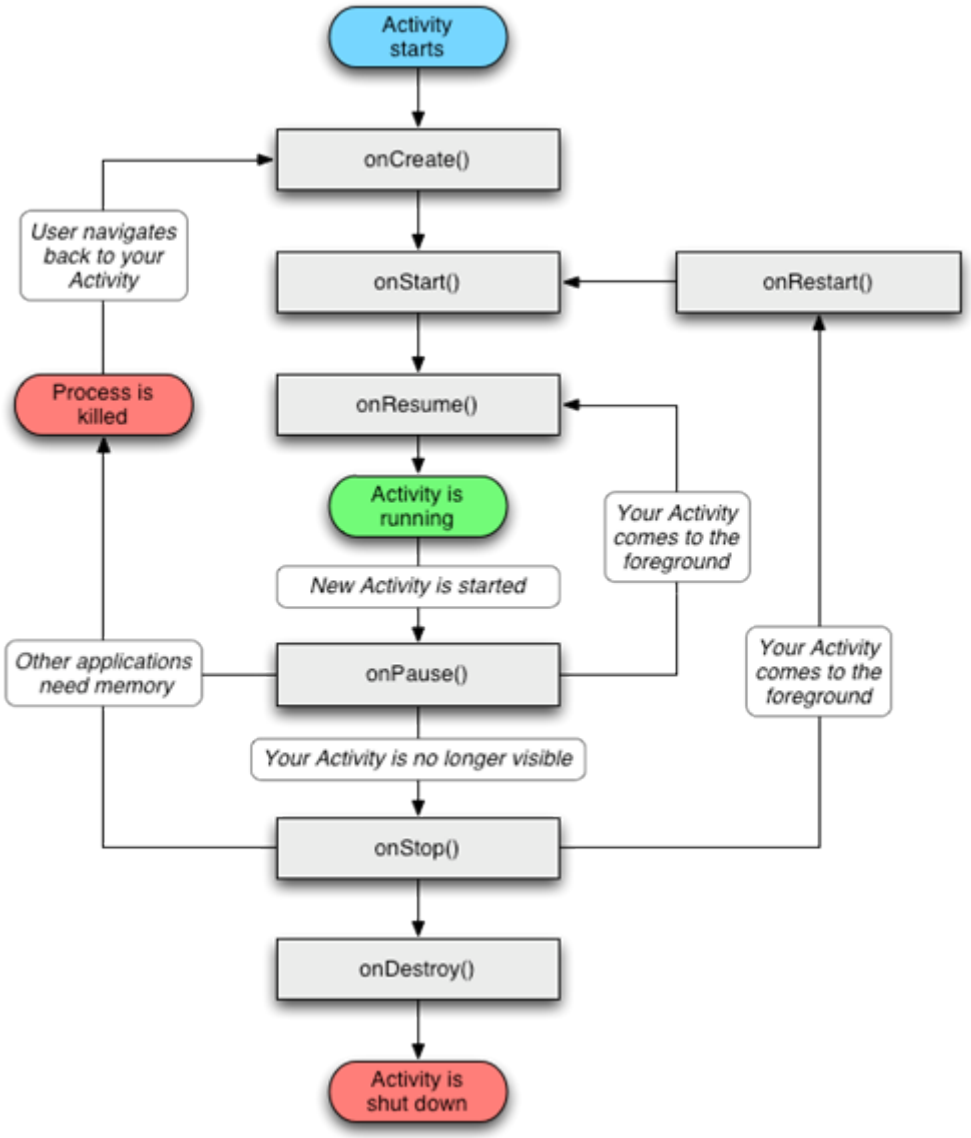
Android Application Overview (cont.)

- Application Lifecycle
  - To free up resources, processes are being killed based on their priority:
    - Critical Priority: foreground (active) processes
      - Foreground activities; components that execute an `onReceive` event handler; services that are executing an `onStart, onCreate,` or `onDestroy` event handler.
    - High Priority: visible (inactive) processes and started service processes
      - Partially obscured activity (lost focus); services started.
    - Low Priority: background processes
      - Activities that are not visible; activities with no started service

Application Components and Lifecycle (cont.)

- Activity Lifecycle:
  - Activities are managed as an activity stack (LIFO collection)
  - Activity has four states:
    - Running: activity is in the foreground
    - Paused: activity has lost focus but it is still visible
    - Stopped: activity is not visible (completely obscured by another activity)
    - Inactive: activity has not been launched yet or has been killed.

Application Components and Lifecycle (cont.)



Source: http://code.google.com/android/reference/android/app/Activity.html#ActivityLifecycle

Outline

- Introduction
- Application Components and Lifecycle
- User Interfaces
- Binding applications and their components
- Data Storage
- Background Services
- Location-Based Services
- Accessing Android’s Sensors
- References

User Interfaces

- Views
  - The basic UI component
  - Responsible for drawing and event handling
  - Define your View through:
    - Layout Resources (i.e. defined in `main.xml` file):

```
<ListView
    android:id="@+id/myListView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
/>
```

      From your Activity class code:

```
setContentView(R.layout.main);
ListView myListView =
(ListView)findViewById(R.id.myListView);
```

    - Inside your code:

```
ListView myListView = new ListView(this);
setContentView(myTextView);
```

  - View Gallery: http://code.google.com/android/reference/view-gallery.html

User Interfaces (cont.)

- Layouts
  - Specify the position of child views (controls) on the screen
  - Common Layout Objects:
    - `FrameLayout`: all child views are pinned to the top left corner of the screen
    - `LinearLayout`: each child view is added in a straight line (vertically or horizontally)
    - `TableLayout`: add views using a grid of rows and columns
    - `RelativeLayout`: add views relative to the position of other views or to its parent.
    - `AbsoluteLayout`: for each view you add, you specify the exact screen coordinate to display on the screen
  - More info: http://code.google.com/android/devel/ui/layout.html

User Interfaces (cont.)

- Implement layouts in XML using external resources:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <EditText
      android:id="@+id/myEditText"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text=""
    />
    <ListView
      android:id="@+id/myListView"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
    />
</LinearLayout>
```

User Interfaces (cont.)

- Menus
  - Concerned about having to much functionality on the screen => use menus
  - Three menu types:
    - **Icon Menu**: appears at the bottom of the screen when the user presses the Menu button. It can display icons and text for up to six menu items.
    - **Expanded Menu**: displays a scrollable list of menu items not previously displayed in the icon menu.
    - **Submenu**: displayed as a floating window.
  - More info: http://code.google.com/android/reference/android/view/Menu.html

Outline

- Introduction
- Application Components and Lifecycle
- User Interfaces
- Binding applications and their components
- Data Storage
- Background Services
- Location-Based Services
- Accessing Android’s Sensors
- References

Binding applications and their components

- Intents
  - Specify what intentions you have in terms of a specific action being performed
  - Launch Activities
    - Transition between the activities of your application
    - Explicitly (using `new Intent(current_application_context, new_activity_to_start);`):

```
Intent newIntent = new Intent(this,
  OtherActivity.class);
startActivity(newIntent); //OtherActivity will become
  visible
```

    - Implicitly (using `new Intent(action_to_perform,data_to_perform_action_on);`):

```
Intent newIntent = new Intent(Intent.ACTION_DIAL,
                              Uri.parse("tel:12345"));
startActivity(newIntent);
```

Binding applications and their components (cont.)

- Intents
  - Broadcast Events
    - Broadcast messages between components (`sendBroadcast(newIntent)` – where `newIntent` is the intent you want to broadcast)
    - Listen for broadcasts using Broadcast Receivers
      - Register a Broadcast Receiver in your application manifest:

```
<receiver android:name=“.YourBroadcastReceiver”>
   <intent-filter>
     <action android:name=
                    “edu.fau.csi.action.NEW_ACTION”>
   </intent-filter>
</receiver>
```

  - More info: http://code.google.com/android/reference/android/content/Intent.html

Binding applications and their components (cont.)

- Adapters
  - Bind data to user interface views
  - Responsible for creating a view for each item in the data set and providing access to the data
  - Example of native adapter:
    - `ArrayAdapter`: binds Adapter views to an array of objects.

```
ArrayList<String> myStringArray = new ArrayList<String>();
ArrayAdapter<String> myArrayAdapter = new
      ArrayAdapter<String>(getApplicationContext(),
      android.R.layout.simple_list_item_1, myStringArray);
myListView.setAdapter(myArrayAdapter);
```

Outline

- Introduction
- Application Components and Lifecycle
- User Interfaces
- Binding applications and their components
- Data Storage
- Background Services
- Location-Based Services
- Accessing Android’s Sensors
- References

Data Storage

- Different techniques for saving data:
  - **Shared Preferences**: lightweight mechanism to store a known set of key-value pairs
    - Useful for saving user preferences, application settings, and user interface state

```
SharedPreferences mySharedPreferences =
     getSharedPreferences(“myPreferences”,
                          Activity.MODE_PRIVATE);
SharedPreferences.Editor editor =
     mySharedPreferences.edit();
editor.putString(“textValue”, “Empty”);
editor.commit();
…
SharedPreferences mySharedPreferences =
     getSharedPreferences(“myPreferences”,
                          Activity.MODE_PRIVATE);
String stringPreference =
     mySharedPreferences.getString(“textValue”,””);
```

Data Storage (cont.)

- Different techniques for saving data:
  - **SQLite Databases**: relational database library for storing and managing complex data
    - Results from database queries are stored in `Cursors`
    - Look at `SQLiteOpenHelper` and `Cursor` class
    - More Info: http://www.sqlite.org/
  - **Files**: you can create, write, and read files from the local storage or external media (SD Cards)
    - Look at `FileOutputStream`, `FileInputStream`, and `Resources` classes.

Data Storage (cont.)

- Content Providers
  - Mechanism for sharing data between applications by abstracting the underlying data source
  - Access is handled through a URI model
  - Native Android Content Providers
    - `Browser`
    - `Contacts`
      - Get a `Cursor` for every person in your contact database:

```
Cursor contactCursor =
  getContentResolver().query(People.CONTENT_URI,
  null, null, null);
```

    - `MediaStore`
    - `...`

Outline

- Introduction
- Application Components and Lifecycle
- User Interfaces
- Binding applications and their components
- Data Storage
- Background Services
- Location-Based Services
- Accessing Android’s Sensors
- References

Background Services

- Services run in the background
- Primarily used for:
  - Updating Content Providers
  - Firing Intents
  - Triggering Notifications
  - Any operation that does not necessitate user interaction (i.e. networking, MP3 playback)
- For intensive and/or blocking operations, the service should be run in its own thread

Background Services (cont.)

- Creating and Controlling Services
  - Create a Service:
    - Extend the `Service` class; override specific methods (such as `onCreate`, `onStart`, `onBind`, etc).
  - Start and stop a Service:
    - Use the `startService` method from inside your current `Activity` class
    - Use the `stopService` method from inside your current `Activity` class
- If the phone becomes inactive while you have services running, those services will not work properly (freeze)
  - Stop your phone from going into sleep mode
    - Use WakeLocks (with care) (http://code.google.com/android/reference/android/os/PowerManager.html)

Outline

- Introduction
- Application Components and Lifecycle
- User Interfaces
- Binding applications and their components
- Data Storage
- Background Services
- Location-Based Services
- Accessing Android’s Sensors
- References

Location-Based Services

- Selecting a Location Provider
  - To determine your current location, Android can use several technologies (or Location Providers)
    - GPS Provider – determines location using satellites
    - Network Provider – determines location using cell towers and Wi-Fi access points
  - Each provider has a set of criteria (power consumption, cost, response time, accuracy, etc.) under which it may be used

Location-Based Services (cont.)

- Finding you location

```
LocationManager locationManager =
  (LocationManager)getSystemService(Context
  .LOCATION_SERVICE);
Location location =
  locationManager.getLastKnownLocation(Loca
  tionManager.GPS_PROVIDER);
```

Location-Based Services (cont.)

- Geocoding
  - Forward Geocoding: finds latitude and longitude of an address
    - Use method `getFromLocationName` from the `Geocoder` class
  - Reverse Geocoding: finds the street address for a given latitude and longitude
    - Use method `getFromLocation` from the `Geocoder` class

Location-Based Services (cont.)

- Map-Based Activities
  - Classes that support Android maps:
    - `MapView`: a view which displays a map. Used within a `MapActivity`
    - `MapActivity`: manages all that is required for displaying a map
    - `Overlay`: used for annotating maps (i.e. drawing text on the map)
    - `MapController`: used for panning and zooming
    - `MyLocationOverlay`: used to display the current position and orientation of the device

Location-Based Services (cont.)

- Using the default MapView centered at the current user position:

Outline

- Introduction
- Application Components and Lifecycle
- User Interfaces
- Binding applications and their components
- Data Storage
- Background Services
- Location-Based Services
- Accessing Android’s Sensors
- References

Accessing Android's Sensors

- The `SensorManager` is used to manage the sensor hardware available on an Android device:

```
SensorManager sensorManager =
 (SensorManager)getSystemService(Context.SENSOR_
SERVICE);
```

- Monitoring changes in sensor values:

```
SensorListener sensorListener = new
SensorListener() {
   public void onSensorChanged(int
 sensor, float[] values) { … }
}
```

  - The values depend on the type of sensor (i.e. accelerometer, light, magnetic field, temperature, proximity)

Outline

- Introduction
- Application Components and Lifecycle
- User Interfaces
- Binding applications and their components
- Data Storage
- Background Services
- Location-Based Services
- Accessing Android’s Sensors
- References

References

- Main Website: http://code.google.com/android/
- Recommended Reading:
  - Reto Meier, "*Professional Android Application Development*", Wrox Programmer to Programmer
  - Mark, L. Murphy, "*The Busy Coder's Guide to Android Development*", CommonsWare
- Android Discussion Groups: http://code.google.com/android/groups.html
- Publish Applications: Android Market, AndAppStore, Handango, SlideME.