Android
Android

- Mobile operating system developed by Google
- A complete stack
  - OS, framework
  - A rich set of applications
    - Email, calendar, browser, maps, text messaging, contacts, camera, dialer, music player, settings and others
- Based on the Linux kernel
- Open source under the Apache 2 license
Dalvik Virtual Machine ( <= 4.4)
Android Runtime (>= 5.0)
ART

- Introduces the use of ahead-of-time (AOT) compilation
- Compiles entire applications into native machine code upon their installation
- Eliminates Dalvik's interpretation and trace-based JIT compilation
- Improves overall execution efficiency and reduces power consumption
- Brings faster execution of applications, improved memory allocation and garbage collection
Android 7 (Nougat)
Set up

• Set up your development environment
  — Android Studio (v 2.2.1)
    • Uses Gradle
• Set up AVDs and devices for testing
  — Create Android Virtual Devices (AVDs)
  — Connect real devices
  — Use external emulators (Genymotion)
Class Overview

Represents a push-button widget. Push buttons can be pressed, or clicked, by the user to perform an action.

A typical use of a push-button in an activity would be the following:

```java
public class MyActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.content_layout_id);

        final Button button = (Button) findViewById(R.id.button_id);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                // Perform action on click
            }
        });
    }
}
```

However, instead of applying an `OnClickListener` to the button in your activity, you can assign a method to your button in the XML layout, using the `android:onClick` attribute. For example:

```xml
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:onClick="mMyButtonHandler" />
```
Many different versions

<table>
<thead>
<tr>
<th>Version</th>
<th>Code name</th>
<th>Release date</th>
<th>API level</th>
<th>DVM/ART</th>
<th>Distribution</th>
<th>First devices to run version</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.0</td>
<td>Nougat</td>
<td>August 22, 2016</td>
<td>24</td>
<td>Unknown</td>
<td>&quot;less than 0.1%&quot;</td>
<td>Nexus 5X, Nexus 6P</td>
</tr>
<tr>
<td>6.0</td>
<td>Marshmallow</td>
<td>October 5, 2015</td>
<td>23</td>
<td>Unknown</td>
<td>18.7%</td>
<td>Nexus 5X, Nexus 6P</td>
</tr>
<tr>
<td>5.1</td>
<td>Lollipop</td>
<td>March 9, 2015</td>
<td>22</td>
<td>Unknown</td>
<td>21.9%</td>
<td>Android One</td>
</tr>
<tr>
<td>5.0</td>
<td>Lollipop</td>
<td>November 3, 2014</td>
<td>21</td>
<td>ART 2.1.0</td>
<td>13.1%</td>
<td>Nexus 6</td>
</tr>
<tr>
<td>4.4</td>
<td>KitKat</td>
<td>October 31, 2013</td>
<td>19</td>
<td>ART 1.6.0</td>
<td>27.7%</td>
<td>Nexus 5</td>
</tr>
<tr>
<td>4.3</td>
<td>Jelly Bean</td>
<td>July 24, 2013</td>
<td>18</td>
<td>Unknown</td>
<td>2.3%</td>
<td>Nexus 7 2013</td>
</tr>
<tr>
<td>4.2</td>
<td>Jelly Bean</td>
<td>November 13, 2012</td>
<td>17</td>
<td>Unknown</td>
<td>7.7%</td>
<td>Nexus 4, Nexus 10</td>
</tr>
<tr>
<td>4.1</td>
<td>Jelly Bean</td>
<td>July 9, 2012</td>
<td>16</td>
<td>Unknown</td>
<td>5.6%</td>
<td>Nexus 7</td>
</tr>
<tr>
<td>4.0</td>
<td>Ice Cream Sandwich</td>
<td>December 16, 2011</td>
<td>15</td>
<td>Unknown</td>
<td>1.4%</td>
<td>Galaxy Nexus</td>
</tr>
<tr>
<td>2.3</td>
<td>Gingerbread</td>
<td>February 9, 2011</td>
<td>10</td>
<td>DVM 1.4.0</td>
<td>1.5%</td>
<td>Nexus S</td>
</tr>
<tr>
<td>2.2</td>
<td>Froyo</td>
<td>May 20, 2010</td>
<td>8</td>
<td>Unknown</td>
<td>0.1%</td>
<td>Droid 2</td>
</tr>
</tbody>
</table>

https://en.wikipedia.org/wiki/Android_(operating_system)
SDK Manager
Development

- **Create** your application
  - Source code, resource files, and Android manifest file
- **Build** and run your application
- **Test** your application
- **Publish** your application
Packaging

• An application is a single APK (application package) file
• An APK file roughly has three main components
  – Executable code: all your compiled Java source code
  – Resources: everything that is not code (images, audio/video clips, XML files describing layouts, language packs, and so on)
  – Native libraries: e.g. C/C++ libraries
  – Manifest
Manifest

• Permissions the application requires
• Minimum API Level required by the application
• Hardware and software features used or required by the application
• Libraries the application needs to be linked with
• … and more
Android Security Model

- Application Isolation
- Application Signing
- Filesystem Isolation
Security model

• Every application runs in its own Linux process
• Each process has its own virtual machine (VM)
  — An application runs in isolation with respect to other applications
• Different applications can not access each other’s data
  — Permissions required
<table>
<thead>
<tr>
<th>Key terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Activity</strong></td>
</tr>
<tr>
<td>Component for composing UI</td>
</tr>
<tr>
<td><strong>Service</strong></td>
</tr>
<tr>
<td>Component that runs on the background</td>
</tr>
<tr>
<td><strong>Intent</strong></td>
</tr>
<tr>
<td>Message element that sends actions and data to components</td>
</tr>
<tr>
<td><strong>Intent filter</strong></td>
</tr>
<tr>
<td>Defines components by setting receiving intents</td>
</tr>
<tr>
<td><strong>Broadcast receiver</strong></td>
</tr>
<tr>
<td>Receives/responds to a certain broadcast (e.g., low battery)</td>
</tr>
<tr>
<td><strong>Content provider</strong></td>
</tr>
<tr>
<td>Provides standardized interfaces for sharing data among applications</td>
</tr>
<tr>
<td><strong>Notification</strong></td>
</tr>
<tr>
<td>Notifies events to users</td>
</tr>
</tbody>
</table>
Android through MVC

View and related classes

Controller
- Activity
- Service
- Broadcast receiver

Model
- SQLite
- Files and content providers
Applications and activities

• Any application can start another application’s component

• An application cannot directly activate a component from another application
  – To activate a component in another application, you must deliver a message to the system that specifies your intent to start a particular component
  – The system then activates the component for you
Starting activities

• The MAIN/LAUNCHER intent filter identifies the app's starting activity
• Nothing forces each app to have one and only one starting activity
• The launcher screen would have different icons for the same program
First App
Add an Activity to Mobile

- Add No Activity
- Basic Activity
- Empty Activity
- Fullscreen Activity
- Google AdMob Ads Activity
- Google Maps Activity
- Login Activity
- Master/Detail Flow
- Navigation Drawer Activity
- Scrolling Activity
- Settings Activity
- Tabbed Activity

[Buttons: Cancel, Previous, Next, Finish]
My Application

Text: "Hello World!"
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="luc.polimi.it.myapplication.MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"/>
</RelativeLayout>
Example
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="luc.polimi.it.myapplication">

<application>
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
</manifest>
<resources>
    <string name="app_name">My First App</string>
    <string name="edit_message">Enter a message</string>
    <string name="button_send">Send</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
</resources>
public class Main extends Activity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}

R Class

• When your application is compiled
  – aapt generates class R
  – It contains resource ids for all resources in res directory
• There is a subclass for each type of resources
  – For example, R.layout for all layout resources
  – For each resource of a type, there is a static integer
    • For example, R.layout.main
  – This integer is the resource id that you can use to retrieve your resource
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_action_search=0x7f020000;
        public static final int ic_launcher=0x7f020001;
    }
    public static final class id {
        public static final int button=0x7f060000;
        public static final int tf=0x7f060001;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int button_label=0x7f040004;
        public static final int hello=0x7f040001;
        public static final int output=0x7f040003;
        public static final int title_activity_main=0x7f040002;
    }
    public static final class style {
        public static final int AppTheme=0x7f050000;
    }
}
Activities

- Each activity can start another activity
  - Each time a new activity starts, the previous activity is stopped
- The system preserves the activities in a LIFO stack
  - The new activity is pushed on top of the stack and takes the focus
Activity launched

onCreate()

onStart()

onResume()

Activity running

Another activity comes into the foreground

onPause()

onStop()

The activity is no longer visible

onDestroy()

App process killed

User navigates to the activity

Apps with higher priority need memory

User returns to the activity

User navigates to the activity

Activity shut down
Three different phases

• Entire lifetime
  – Between onCreate() and onDestroy()
  – Setup of global state in onCreate()
  – Release remaining resources in onDestroy()

• Visible lifetime
  – Between onStart() and onStop()
  – Maintain resources that have to be shown to the user

• Foreground lifetime
  – Between onPause() and onResume()
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Killable?</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>onCreate()</code></td>
<td>Called when the activity is first created. This is where you should do all of your normal static set up: create views, bind data to lists, etc. This method also provides you with a Bundle containing the activity's previously frozen state, if there was one. Always followed by <code>onStart()</code></td>
<td>No</td>
<td><code>onStart()</code></td>
</tr>
<tr>
<td><code>onRestart()</code></td>
<td>Called after your activity has been stopped, prior to it being started again.</td>
<td>No</td>
<td><code>onStart()</code></td>
</tr>
<tr>
<td><code>onStart()</code></td>
<td>Called when the activity is becoming visible to the user. Followed by <code>onResume()</code> if the activity comes to the foreground, or <code>onStop()</code> if it becomes hidden.</td>
<td>No</td>
<td><code>onResume()</code> or <code>onStop()</code></td>
</tr>
<tr>
<td><code>onResume()</code></td>
<td>Called when the activity will start interacting with the user. At this point your activity is at the top of the activity stack, with user input going to it. Always followed by <code>onPause()</code></td>
<td>No</td>
<td><code>onPause()</code></td>
</tr>
<tr>
<td><code>onPause()</code></td>
<td>Called when the system is about to start resuming a previous activity. This is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, etc. Implementations of this method must be very quick because the next activity will not be resumed until this method returns. Followed by either <code>onResume()</code> if the activity returns back to the front, or <code>onStop()</code> if it becomes invisible to the user.</td>
<td>Pre-HONEYCOMB</td>
<td><code>onResume()</code> or <code>onStop()</code></td>
</tr>
<tr>
<td><code>onStop()</code></td>
<td>Called when the activity is no longer visible to the user, because another activity has been resumed and is covering this one. This may happen either because a new activity is being started, an existing one is being brought in front of this one, or this one is being destroyed. Followed by either <code>onRestart()</code> if this activity is coming back to interact with the user, or <code>onDestroy()</code> if this activity is going away.</td>
<td>Yes</td>
<td><code>onRestart()</code> or <code>onDestroy()</code></td>
</tr>
<tr>
<td><code>onDestroy()</code></td>
<td>The final call you receive before your activity is destroyed. This can happen either because the activity is finishing (someone called <code>finish()</code> on it) or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the <code>isFinishing()</code> method.</td>
<td>Yes</td>
<td>nothing</td>
</tr>
</tbody>
</table>
Intents

• Allow for late binding between components
  – Activities, Services, Broadcast receivers
• Indirect communication
• Reuse existing and installed applications
• Two possible types
  – Explicit: The target receiver is specified through the Component Name
  – Implicit: The target is specified by data type/names. The system chooses the receiver that matches the request
Simple example

Activity A

1

Intent

startActivity()

Android System

2

Intent

onCreate()

Activity B

3
Intent definition

• Component Name (optional): the receiver of the intent
• Action name: the action to be performed
  — Predefined actions exist, the programmer can define new ones
• Data: information exchanged between caller and callee
• Category: the kind of component that should handle the Intent
• Extras: additional information in the form of key-value pairs
• Flags: additional information to instruct Android how to launch an activity, and how to treat it after execution
## Actions

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION_CALL</td>
<td>Perform a call to someone specified by the data</td>
</tr>
<tr>
<td>ACTION_EDIT</td>
<td>Provide explicit editable access to the given data</td>
</tr>
<tr>
<td>ACTION_MAIN</td>
<td>Start as a main entry point, does not expect to receive data</td>
</tr>
<tr>
<td>ACTION_PICK</td>
<td>Pick an item from the data, returning what was selected</td>
</tr>
<tr>
<td>ACTION_VIEW</td>
<td>Display the data to the user</td>
</tr>
<tr>
<td>ACTION_SEARCH</td>
<td>Perform a search</td>
</tr>
</tbody>
</table>
Examples of action/data pairs

- **ACTION_VIEW** content://contacts/people/1
  - Display information about the person whose identifier is "1"
- **ACTION_DIAL** content://contacts/people/1
  - Display the phone dialer with the person filled in
- **ACTION_VIEW** tel:123
  - Display the phone dialer with the given number filled in
- **ACTION_DIAL** tel:123
  - Display the phone dialer with the given number filled in.
- **ACTION_EDIT** content://contacts/people/1
  - Edit information about the person whose identifier is "1".
- **ACTION_VIEW** content://contacts/people/
  - Display a list of people, which the user can browse through
Explicit intents

- `startActivity(Intent x)` starts a new activity, and places it on top of the stack
  - The Intent parameter describes the activity we want to execute
  - Often they do not include any other information
    - It is a way for an application to launch internal activities
- `new Intent(Context c, Class c);`
  - Context is a wrapper for global information about an application environment
  - Activity subclasses Context
Explicit intents

Intent intent=new Intent(this, SndAct.class);
startActivity(intent);

Intent intent=new Intent();
ComponentName component=new ComponentName(this, SndAct.class);
intent.setComponent(component);
startActivity(intent);
Intents with results

- Activities can return results
  - startActivityForResult(Intent int, int reqCode);
  - onActivityResult(int reqCode, int resCode, Intent data)

```java
static final int PICK_CONTACT_REQUEST = 1;
// The request code

Intent pickContactIntent = new Intent(Intent.ACTION_PICK,
    Uri.parse("content://contacts"));
pickContactIntent.setType(Phone.CONTENT_TYPE);
// Show user only contacts with phone numbers
startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST);
```
Moreover

```java
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // Check which request we're responding to
    if (requestCode == PICK_CONTACT_REQUEST) {
        // Make sure the request was successful
        if (resultCode == RESULT_OK) {
            // The user picked a contact.
            // The Intent's data Uri identifies which
            // contact was selected.

            // Do something with the contact here
        }
    }
}
```
setResult()

• `void setResult(int resultCode, Intent data)`
• The result is delivered to the caller component only after invoking the `finish()` method!
Implicit intents

• Target component is not named
  – component name is left blank
• When Intent is launched, Android tries to find an activity that can answer it
  – If at least one is found, then that activity is started!
  – If more than one matching activity is found, the user is prompted to make a choice
Example

- Implicit intents are very useful to re-use code and to launch external applications
  - More than a Component can match the Intent request

```
Intent i = new Intent(android.content.Intent.ACTION_VIEW,
                      Uri.parse("http://www.polimi.it"));
startActivity(i);
```
The action specified in the Intent must match one of the actions listed in the filter

- Fail if filter does not specify any action
- Success if intent does not specify an action but the filter contains at least one action

```xml
<intent-filter>
  <action android:name="android.intent.action.EDIT" />
  <action android:name="android.intent.action.VIEW" />
  ...
</intent-filter>
```
Category

• Every category in the Intent must match a category of the filter
  – If the category is not specified in the Intent
    • Android assumes it is DEFAULT
    • The filter must include this category to handle the intent

<intent-filter>
  <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
Data

• The URI of the intent is compared with the parts of the URI mentioned in the filter
  — Both URI and type are compared

<intent-filter>
  <data android:mimeType="audio/*" android:scheme="http"/>
  <data android:mimeType="video/mpeg" android:scheme="http"/>
</intent-filter>
Example

```xml
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

```java
public class Main extends Activity {
    public final static String EXTRA_MESSAGE = "luc.polimi.first.MESSAGE";

    public void sendMessage(View view) {
        Intent intent = new Intent(this, DisplayMess.class);
        EditText eText = (EditText) findViewById(R.id.edit_message);
        String message = eText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
    }
}
```
Another activity

<RelativeLayout
android:layout_width="match_parent"
android:layout_height="match_parent">

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_centerVertical="true"
android:text="@string/hello_world"
tools:context=".DisplayMess" />

</RelativeLayout>
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Get the message from the intent
    Intent intent = getIntent();
    String message = intent.getStringExtra(Main.EXTRA_MESSAGE);

    // Create the text view
    TextView textView = new TextView(this);
    textView.setTextSize(40);
    textView.setText(message);

    // Set the text view as the activity layout
    setContentView(textView);
}
Constraining instances

```java
protected void onCreate(Bundle icicle) {
    super.onCreate(icicle);

    instances++;
    setContentView(R.layout.activity_a2);
    tf = (TextView) findViewById(R.id.instanceCount2);

    final Button button = (Button) findViewById(R.id.button2);
    final Intent intent = new Intent(this, A1.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);

    button.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            startActivity(intent);
        }
    });
}
```
FLAG_ACTIVITY_CLEAR_TOP

• If the activity being launched is already running in the current task
  – Instead of launching a new instance of that activity
  – All of the other activities on top of it will be closed
  – The intent will be delivered to the (now on top) old activity as a new Intent

• Consider a task consisting of activities A, B, C, D
  – If D calls startActivity() with an Intent that resolves to activity B
    • C and D will be finished and B receives the given Intent
    • The stack now is: A, B

• Other flags are available