Android

Jaroslav Dytrych

Faculty of Information Technology Brno University of Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole

dytrych@fit.vutbr.cz

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

16 December 2020

Contents

- What is android?
- Android features
- Architecture overview
- Environment setup
- Simple applications
- Android database
- Android Java Services

What is Android?

- Operating system based on Linux and Java
- Developed by Google
- Dalvik instead of standard JVM (Just-In-Time Compilation)
  - replaced by Android Runtime (ART) in Android 5.0
    - compatible, but some Dalvik bytecode may not work (ART is less tolerant)
    - better optimizations (Ahead-of-time (AOT) compilation – during installation), garbage collection, profiling, ...
- Android SDK
  - Compiler
    - The dx (Dexer) tool converts Java class files into a `.dex` (ART/Dalvik Executable) file

      ```
      javac Hello.java
      jar cvf hello.jar Hello.class
      dx --dex --output=helloApp.jar hello.jar
      ```
    - `helloApp.jar` will contain `classes.dex`
    - Alternatives:

      ```
      dx --dex --output=classes.dex Hello.class
      dx --dex --output=hello.apk hello.jar
      ```
  - Debugger
  - Emulator
- Google Play (previously Android market)

File types

- Application (`.apk`)
  - basically improved zip archives with the Java code in a file `classes.dex`. This file is parsed by the ART/Dalvik JVM and a cache of the processed `classes.dex` file is stored in the phone's cache.
  - created using IDE, Gradle, ant or dx
  - may contain **assets** (images, XML files, prefilled databases, ...)
- `.odex` (optimized dex) – pre-processed version of an application's `classes.dex` that is execution-ready for ART/Dalvik (AOT compiled code)
  - only a part of application can be odexed for installing of `.apk` and `.odex`
  - in odexed application the `classes.dex` is removed from the APK archive and it does not write anything to the cache
  - Deodexing is basically repackaging of APKs in a certain way, such that they are reassembled into `classes.dex` files. By doing that, all pieces of an application package are put together back in one place, thus eliminating the worry of a modified APK conflicting with some separate odexed parts.

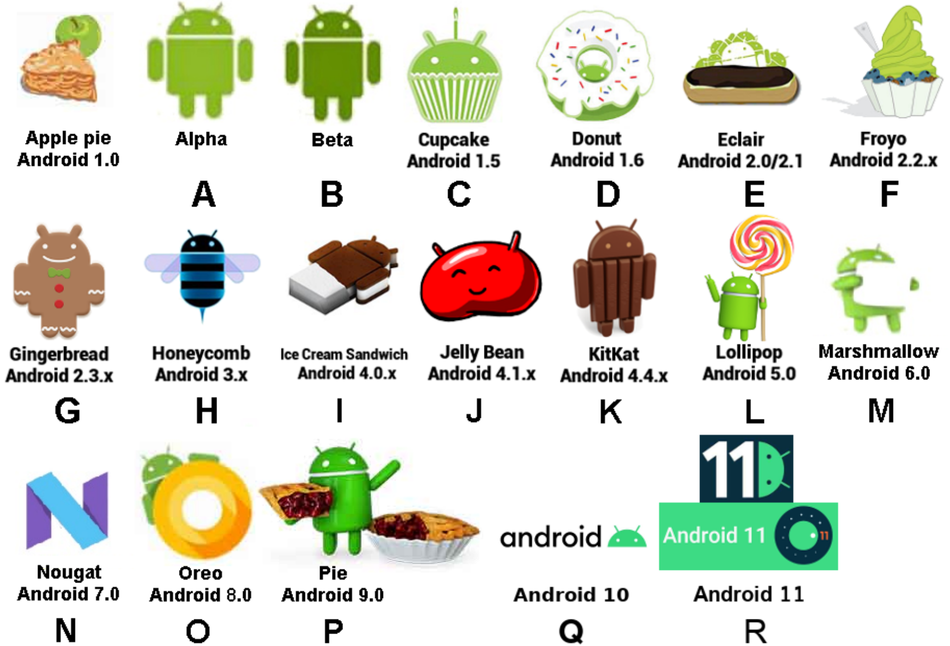
File types

- Of Ahead Time (`.oat`)
  - pre-processed version of an application's `classes.dex`
  - alternative to `.odex`
  - Executable and Linkable Format (ELF)
- Uncompressed DEX (`.vdex`)
  - contains the uncompressed DEX code of the APK, with some additional metadata to speed up verification.
  - Prior Android Oreo (8.0.0) DEX files were embedded in the OAT itself and after Oreo, the transformation performed by dex2oat generates two files.
- (`.art`)
  - contains ART internal representations of some strings and classes listed in the APK, used to speed application startup.

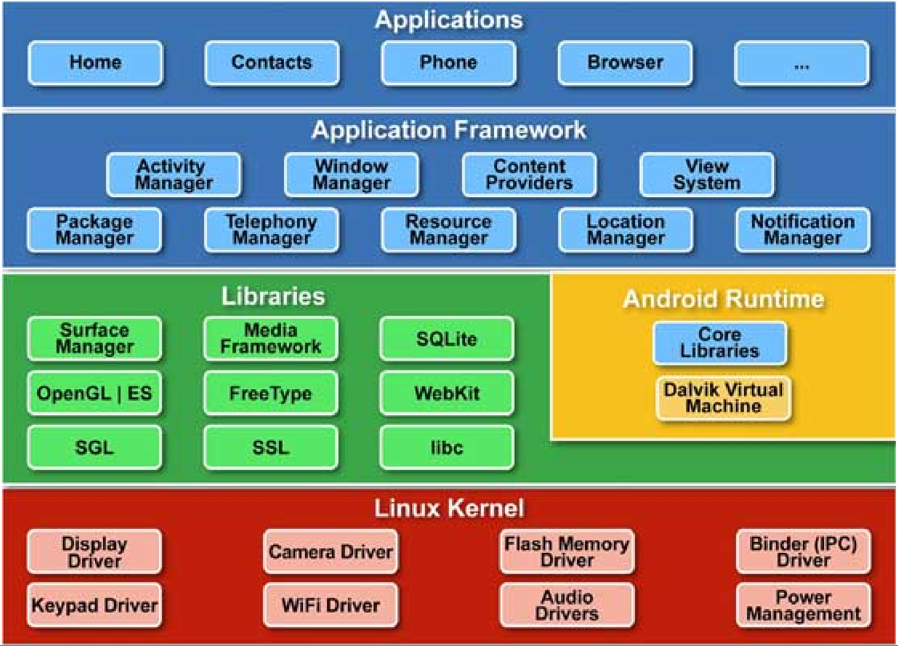
Android features

- Application ecosystem, allowing to easily add and remove applications and publish new features across the entire system.
- Support for all the web technologies, with a browser built on top of the well-established WebKit rendering engine.
- Support for hardware accelerated graphics through OpenGL ES (Embedded Systems).
- Support for all the common wireless mechanisms: GSM, CDMA, UMTS, LTE, Bluetooth, WiFi.

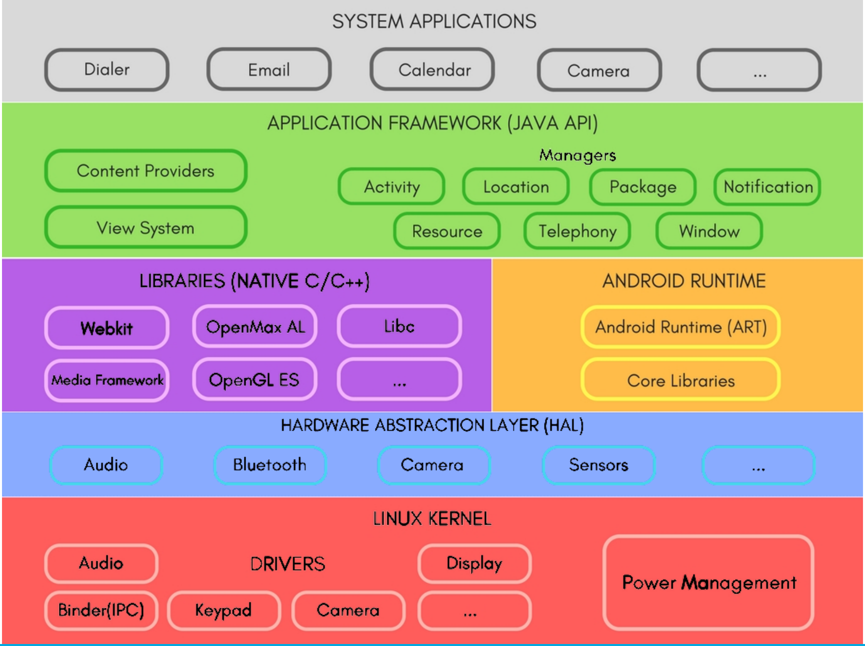
Android versions



Android´s architecture – old



Android's architecture – new



Android's architecture

- Linux kernel contains all the essential hardware drivers like display, camera, keypad etc.
- Hardware Abstraction Layer (HAL) provides abstraction between hardware and rest of the software stack.
- Libraries includes
  - WebKit – a web browser engine,
  - SSL library for the Internet security,
  - graphic and multimedia libraries OpenGL ES, SGL (Scalable Graphics Library) and OpenMAX AL (Open Media Acceleration – Application Layer),
  - FreeType for rendering of fonts,
  - Surface Manager for access to the display subsystem and seamlessly composing of 2D and 3D graphic layers from multiple applications,
  - SQLite for storing of application data (one database per application is expected),
  - ...

Android's architecture

- Application Framework is a set of services that collectively form the environment in which Android applications run.
  - Content Providers – allows applications to publish and share data with other applications.
  - View System – is an extensible set of views used to create application user interfaces.
  - Activity Manager – controls all aspects of the application lifecycle and activity stack.
  - Window Manager – is responsible for managing the list of windows, which windows are visible, ...
  - Package Manager – is the system by which applications are able to find out information about other applications currently installed on the device.
  - Telephony Manager – provides information to the application about the telephony services available on the device.
  - Resource Manager – provides access to non-code embedded resources such as strings, color settings and user interface layouts.
  - Location Manager – provides access to the location services allowing an application to receive updates about location changes.
  - Notifications Manager – allows applications to display alerts and notifications to the user.

Security and permissions

- Unique ID for every application.
  - Applications are separated.
  - Sharing is always explicit.
- Each application started in own process.
- Permissions (for Internet connection, telephone services, camera, . . .)
  - must be declared in Android manifest.
  - Different levels of permissions.
  - User is asked during the installation (up to version 5) or on runtime (From Android 6.0 – API level 23).

Android components

- Activities
  - basically regular applications.
  - Activity is a screen of user interface (e.g. login activity).
- Fragments
  - parts of activities (created from views),
  - often parts of user interface from 3rd parties.
- Views and layout manager
  - buttons, check boxes, text fields, grid layout, linear layout, relative layout, . . .
- Intents
  - asynchronous messages
  - between activities
  - implicit (as open the browser) and explicit (data transfer).
- Services
  - background processes like music players.
- Broadcast receivers
  - receives intents.
- Content providers
  - access to the application data and sharing.

Environment setup

- Android SDK
  - Command-line programs
  - Dalvik/ART executables
- Android development tools
  - Android Studio (Official IDE for Android based on IntelliJ IDEA)
  - Eclipse plugins – not supported and deprecated (`https://dl-ssl.google.com/android/eclipse/`)
- Android virtual devices
  - Virtual devices manager
- ADB (Android Debug Bridge)
  - command-line tool that lets you communicate with a device
  - client – sends commands
  - daemon – runs commands on a device (background process on each device)
  - server – manages communication between the client and the daemon (background process on your development machine on port 5037)

Demo – SDK manager

Android debug bridge

- `adb install <path-to-apk>`
    - install an application
- `push <local> <remote>`
    - copy a file or directory and its sub-directories to the device
- `pull <remote> <local>`
    - copy a file or directory and its sub-directories from the device
- `adb forward <local-port> <remote-port>`
    - e.g. VirtualBox
    - example: `adb forward tcp:6100 tcp:7100`
- `adb shell`
    - starts shell in Android environment in the target device
- . . .

`app`

- `build` .... (build outputs)
- `src/main`
  - `assets` .... (images etc.)
  - `res` .... (application resources)
  - `java` .... (src)
  - `AndroidManifest.xml`
- `app.iml` ... (module IntelliJ IDEA project model)
- `build.gradle` .... (Gradle script)

`build` .... (build cache etc.)
`gradle` .... (Gradle wrapper)
`AppName.iml` .... (IntelliJ IDEA project model)
`build.gradle` .... (Gradle script)
`gradlew` .... (Gradle start up script)
`...`

Assets vs. resources

- Assets are files to be deployed with application (e.g. images).
- Resources are application resources, such as drawable files, layout files and UI string.
    - Often XML files.

- File `AndroidManifest.xml`
- unique identifier for the application (Java package name),
- the level of the Android API that the application requires – SDK version (here deprecated – should be in `app/build.gradle`)
  - `minSdkVersion` – minimum API Level required for the application to run,
  - `targetSdkVersion` – API Level that the application targets (used for compilation),
  - `maxSdkVersion` – maximum API Level on which the application is designed to run.
- used libraries,
- permissions setup,
- components of the application, which include the activities, services, content providers and broadcast receivers,
- entry-point specification,
- ...

Example AndroidImage

- When developing apps that support multiple API versions, you may want a standard way to provide newer features on earlier versions of Android or gracefully fall back to equivalent functionality. Rather than building code to handle earlier versions of the platform, you can leverage these libraries to provide that compatibility layer. In addition, the Support Libraries provide additional convenience classes and features not available in the standard Framework API for easier development and support across more devices.

- Dependency (Android Studio)
  - `build.gradle` (Project)
    ```
    allprojects {
      repositories {
        google()
        ...
    ```
  - `build.gradle` (Module)
    ```
    dependencies {
      api 'com.android.support:support-v4:28.0.0'
    ```
- Installation (Eclipse)
  1. Install Android Support Repository in the SDK manager (Extras).
  2. File – Import.
  3. Existing Android Code into Workspace.
  4. Browse to the SDK installation directory
     `android-sdks/extras/android/support/v7/appcompat`
  5. Finish.
  6. In the imported library project, expand the `libs/` folder, right-click each `.jar` file and select Build Path – Add to Build Path.
  7. Go to the your project properties – Android category.
  8. Add (in the part Library).
  9. Select the project and confirm.

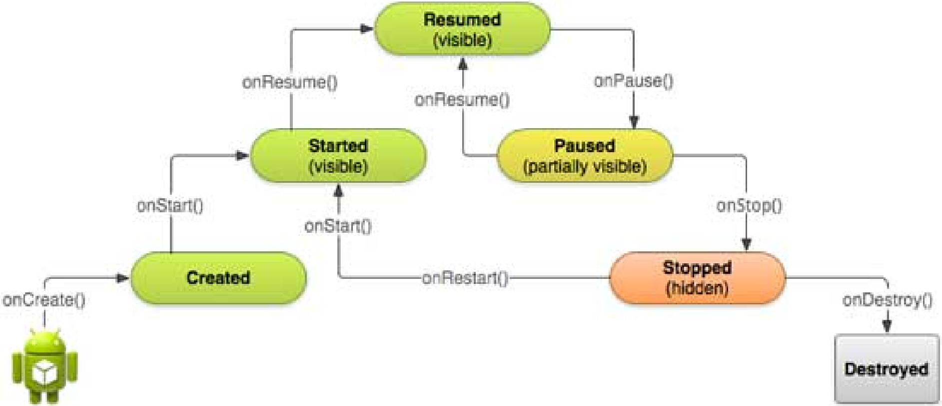
AndroidX

- Android Support Library is deprecated.
- AndroidX replaces the original support library APIs with packages in the androidx namespace.
- Only the package and Maven artifact names changed; class, method, and field names did not change.
- Migration
  - Before you migrate, bring your app up to date. It is recommended to update your project to use the final version of the support library: version 28.0.0. This is because AndroidX artifacts with version 1.0.0 are binary equivalent to the Support Library 28.0.0 artifacts.
  - With Android Studio 3.2 and higher, you can migrate an existing project to AndroidX by selecting Refactor – Migrate to AndroidX from the menu bar.

Activity

- Activity is a single screen of the user interface of an application.
- Activity supports advertising.
- When an activity starts a new activity, the latter replaces the former on the screen and is pushed on the back stack which holds the last used activities, so when the user is done with the newer activity, it can easily go back to the previous one
  - button Back, gesture or calling of method `finish()`

Android activity



Activity lifecycle

- Started
  - Created a new Linux process, allocated new memory for the new UI objects, and set up the whole screen (visible).
- Running (Resumed)
  - The activity is on the foreground and has focus.
- Paused
  - The activity is still visible on the screen but no longer has focus. It can be destroyed by the system under very heavy memory pressure.
- Stopped
  - The activity is no longer visible on the screen. It can be killed at any time by the system.
  - It is in the memory for quick restart.

- Each activity extends class `Activity` with callback methods.

Android resources

- Simple values
  - strings (`/res/values/strings.xml`),
  - constants used in program.
- Layouts
  - layout descriptions for activities and fragments.
- Styles and themes
  - define appearance (like Holo light with dark action bar).
- Animations
  - define animations in XML for the property animation API.
- Menus
  - properties of entries for a menu.

Android resources

- `R.java`
  - ID's are generated there,
  - in the XML files we define a names which is mapped into number constants in the generated `R.java`

    ```
    public static final int
    activity_horizontal_margin=0x7f040000;
    ```
  - constants then can be used to create an objects.
- Android Gradle plugin 3.6 and higher includes support for the Maven Publish Gradle plugin, which allows you to publish build artifacts to an Apache Maven repository. Additionally, Android Gradle plugin has made significant performance improvement for annotation processing/KAPT for large projects. This is caused by AGP now generating R class bytecode directly, instead of .java files.
  - There is no `R.java` from Android Studio 3.6.

Resources directory

- Resources are stored in `/res` folder
  - `anim/` – animation definitions,
  - `color/` – color definitions,
  - `drawable` – subdirectories with images (`ldpi`, `mdpi`, `hdpi`, `xhdpi`),
  - `layout/` – activity layout description,
  - `menu/` – menu layout description,
  - `raw/` – left untouched,
  - `values/` – strings, integers, arrays, etc.,
  - `xml/` – arbitrary XML files.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
  <solid android:color="#FF1A47"/>
  <stroke android:width="3dp"
          android:color="#0FECFF"/>
  <padding android:left="5dp"
           android:top="5dp"
           android:right="5dp"
           android:bottom="5dp"/>
  <corners android:bottomRightRadius="7dp"
           android:bottomLeftRadius="7dp"
           android:topLeftRadius="7dp"
           android:topRightRadius="7dp"/>
</shape>
```

Drawable example (2/2) – custom widget

```xml
<?xml version="1.0" encoding="UTF-8"?>
<android:shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">
  <gradient android:startColor="#31d931"
            android:endColor="#0da40d"
            android:angle="270"/>
  <stroke android:width="2dp" android:color="#ffc0c0c0" />
  <size android:height="12dp" android:width="12dp"/>
</android:shape>
```

Create first Android application

- Define Application name, Project name and Package name.
- Define SDK and theme of an application
  - minimum SDK version, target SDK version
- Configure launcher icon.
- Specify type of the main activity.
- Specify layout.
- Run AVD (Android Virtual Device).

- Control appearance
  - Wrap content
  - Match parent (Fill parent)
- Define layout/widget with unique ID
  - example: `android:id="@+id/my_button"`
  - `@` – XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
  - `+` – this is a new resource name that must be created and added to our resources (in the `R.java` file).
- Create an instance of the view object and capture it from the layout

```
Button myButton = (Button) findViewById(R.id.my_button);
```

- `Button`
- `CheckBox`
- `Password`
- `RadioButton`
- `ToggleButton`
- `RatingBar`
- `Spinner – drop down menu`
- `ProgressBar`
- `Image`
- `Dialog`

Examples AndroidCheckBox, AndroidPassword, AndroidRadio AndroidRatingBar, AndroidSpinner, AndroidProgressBar2

Dialogs

- Embedded
  - Prompt
  - Alert

    `AlertDialog.Builder`
- Custom
  - LayoutInflater – dynamic creation of an instance from XML
  - DialogInterface – defines a dialog-type class
    - `Dialog.class.cast(DialogInterface)`
    - `BUTTON_NEGATIVE, BUTTON_NEUTRAL, BUTTON_POSITIVE`
  - or use `setContentView()`
    - set the activity content to an explicit view (for Dialog)
    - less dynamic

Examples AndroidAlert, AndroidPrompt, AndroidCustomDialog

Layout managers

- `LinearLayout` – aligns all children in a single direction, vertically or horizontally
  - `layout_weight` – size ratio between multiple views
  - can be nested
- `RelativeLayout` – position of each view can be specified as relative to sibling elements (used by designer, easy to break).
- `TableLayout`
- `Listview`
- `GridView`
  - since Android 4.0
- `WebView` – displays web pages
- `ScrollView`

Examples AndroidLinearLayout, AndroidRelativeLayout,
AndroidTableLayout, AndroidListView, AndroidGridView
AndroidWebView, AndroidScrollView

- What are intents?
  - Asynchronous messages which allows Android components to request functionality from other components of the Android system.
  - They're signalling, that an event has occurred.
  - Sent to system via call `startActivity(Intent)`
  - Can switch activities of an application.
  - Instances of class `android.content.Intent`
  - Can be explicit or implicit
    - Explicit – Developer designates the target by its name,
    - Implicit – There is no explicit target for the Intent. The system will find the best target for the Intent by itself, possibly asking the user what to do if there are several matches (which web browser to use).
  - Activities, Services and BroadcastReceivers starts using intents.

Explicit intents

- Explicit intents explicitly defines the component which should be called by the Android system, by using the Java class as identifier.
  - Associative array of values can be sent in the intent

```
Intent i = new Intent(this, ActivityTwo.class);
i.putExtra("Value1", "This is value one for another
    activity");

Bundle extras = getIntent().getExtras();
String value1 = extras.getString("Value1");
```

Example AndroidExplicitIntent

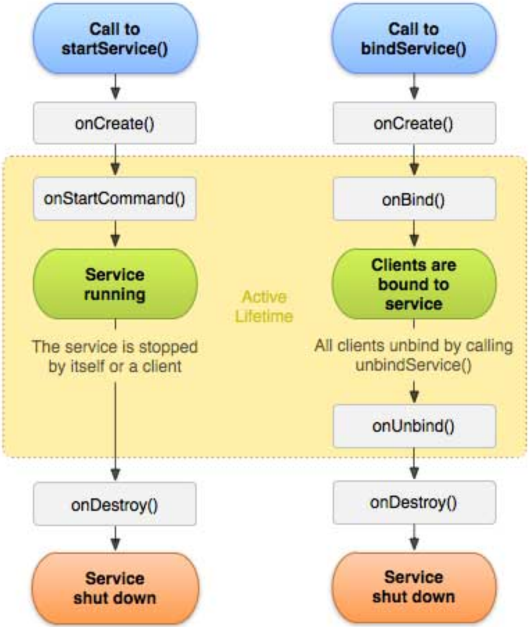
Implicit intents

- Specify the action which should be performed and optionally data for the action.
- If only one handler is registered for an intent, android proceeds with intent directly, otherwise asks user.
  - Like two video players installed . . .

```
Intent i = new Intent(Intent.ACTION_VIEW,
Uri.parse("http://www.fit.vutbr.cz"));
```

- Serves like global service messages.

Example AndroidImplicitIntent



noninteractive

interactive

- `onCreate()` – executed when the service is first created in order to set up the initial configuration.
- `onBind()` – system invokes this method by calling `bindService()` when another component wants to bind with the service (such as to perform RPC).
- `onUnbind()` – called when all clients have disconnected from a particular interface published by the service.
- `onRebind()` – new clients have connected to the service, after it had previously been notified that all had disconnected.
- `onStartCommand()` – executed every time `startService()` is invoked by another component, like an Activity or a BroadcastReceiver. When this method executes, the service is started and can run in the background indefinitely. If you implement this, it is necessary to stop the service by calling `stopSelf()` or `stopService()`.
- `onDestroy()` – the service is no longer used and is being destroyed. Your service should implement this to clean up.

- Service runs in background without user interaction.
- Many predefined services are embedded in system.
- Scope resolved in `AndroidManifest.xml`
  - in a thread,
  - in own process.
- Services runs with higher priority than activities.
- They need to be explicitly declared in manifest.
- Communication via Intents.

  `OnHandleIntent()` – deprecated

  `OnHandleWork()`
- Activity represents context.
- Broadcast receivers reads data from services.

Broadcast receiver

- A component which allows to react on custom and system events.
- Must be registered
  - either statically in android manifest
  - or dynamically via `Context.registerReceiver()`
- `sendBroadcast(Intent)` → `OnReceive(Context, Intent)`
- Intent Filter can be specified.
- System broadcasts
  - `BOOT_COMPLETED`
  - `CONNECTIVITY_CHANGE`

Android Widgets

- Placed on HomeScreen (e.g. battery indicator).
- A Widget runs as a part of the process of its host.
  - This requires that Widgets preserve the permissions of their application.
- Uses `remoteViews` to create user interface.
- Interface for a widget is Broadcast receiver.
- Steps to create a widget
  - Define layout.
  - Create XML – `AppWidgetProviderInfo`
    - Dimensions
    - Update period
    - Category (home screen, the keyguard, or both)
  - Create a `BroadcastReceiver` which is used to build the user interface of the Widget.
  - Enter the Widget configuration in the `AndroidManifest.xml` file.
  - Optionally call the activity from the widget.

Example AndroidWidget

Running example AndroidWidget

- If you have a problem that you have no main activity, in the Android Studio Select menu Run -> Edit Configurations. In the General tab for app, look for Launch Options and select Nothing instead of Default Activity.
- Nothing is visible after installation – it is necessary to hold finger/pointer on empty space on home screen and then add the widget.
- Example is for SDK version less than 26. In the version 26 and higher, it is not working properly.

- Widget challenge 2020/21: who will first fix the example for SDK version 29 will gain 6 bonus points (tutorial is in the references).

Widgets continuation

- `BroadcastReceiver` typically extends `AppWidgetProvider`
- The `AppWidgetProvider` class implements the `onReceive()` method, extracts the required information and calls the following widget lifecycle methods.
- Lifecycle
  - `OnEnabled` – when first instance of a widget is added on homescreen.
  - `OnDisabled` – when the last instance is removed from the homescreen.
  - `OnUpdate` – everytime a message arrives.
  - `OnDeleted` – when any instance of a widget is removed from the screen.

Widget updates

- A Widget gets its data on a periodic timetable.
  - XML configuration updates
  - `AlarmManager` service
- Fixed update interval may be specified in XML configuration.
- The `AlarmManager` allows you to be more resource efficient and to have a higher frequency of updates. To use this approach you define a service and schedule this service via the `AlarmManager` regularly.

Example AlarmActivity

Android Database system

- Uses open source SQLite database.
  - Supports standard SQL queries.
  - Requires only 250 kB of memory.
- Supported datatypes
  - `TEXT` (`String` in Java),
  - `INTEGER` (`Long` in Java),
  - `REAL` (`Double` in Java).
- Requires accessing the file system.
  - Asynchronous processing supported.

- No setup needed.
- Default database file

  `/DATA/APP_NAME/databases/FILENAME`

  - `DATA` is the path which the `Environment.getDataDirectory()` returns.
  - `APP_NAME` is a name of application.
  - `FILENAME` is a name of a database specified in code.

- `Package android.database.sqlite`
- `SQLiteOpenHelper`
  - Manages database updates.
  - `onCreate` – the database is created for the first time (load defaults form assets).
  - `onUpdate` – the database needs to be upgraded. The implementation should use this method to do anything it needs to upgrade to the new schema version.
  - `getReadableDatabase()`
  - `getWritebleDatabase()`
- Database tables should use `_id` as a primary key.

Work with database

- Database installation
  - Check for database existence
    `SQLiteDataBase.openDatabase(path, factory, ...)`
  - If non-existent, copy from Assets.

- Base type for every database.
- Provides methods for working with database
  - `insert`
  - `update`
  - `delete`
  - `execSQL`
  - `rawQuery` – performs raw SQL query.
  - `query` – structured interface for creating a query.

```
Cursor c = getReadableDatabase().rawQuery("select * from
    todo where _id = ?", new String[] { id });
```

Query parameters

- `tableName` – name of the database table.
- `columnNames` – passing `null` returns all collumns.
- `whereClause` – filter, `null` selects all.
- `selectionArgs` – `?`s may be included in the "`whereClause`" – These placeholders will get replaced by the values from the `selectionArgs` array.
- `groupBy` – declare how to group rows.
- `having` – filter, `null` means no filter.
- `orderBy` – how to order the results.

```
return database.query(DATABASE_TABLE, new String[]
    {KEY_ROWID, KEY_CATEGORY, KEY_SUMMARY,
    KEY_DESCRIPTION}, null, null, null, null, null);
```

Cursor

- Query returns cursor object.
- Points to one row of returned array.
- Has several methods
  - `getCount()`
  - `moveToFirst()`
  - `moveToNext()`
  - `getLong(columnIndex)`
  - `getString(columnIndex)`
  - `close()`
  - ...

Example AndroidSQL

Displaying data

- `ListView`
  - Adapters – objects storing collections, that can be displayed.
  - `SimpleCursorAdapter` – defines layout for each row of `ListView`.
  - Cursor to view mapping.
- Direct usage of `ListView` is deprecated - use `ListFragment` instead.
- Define arrays
  - An array which contains the column names.
  - An array which contains the IDs of Views which should be filled with the data.
- `Loader` class
  - loads data from a content provider.

Content Provider

- By default, SQLite is private to application.
- Sharing can be specified via Content Provider.
- Content provider is an application which shares data to other applications, almost allways it is some sort of SQLite database.
- Must be specified in manifest
  - `Android:authorities`
  - Example – contacts
- Subclasses `android.content.ContentProvider`
- Sharing is implicitly on `ContentProvider`
  - Can be turned off.
- Thread safety
  - Keyword `synchronized`

Example AndroidContentProvider

- `CREATE VIRTUAL TABLE data1 USING fts3(content TEXT); // inverted index (Apache Lucene)`
- `CREATE TABLE data2(content TEXT);`

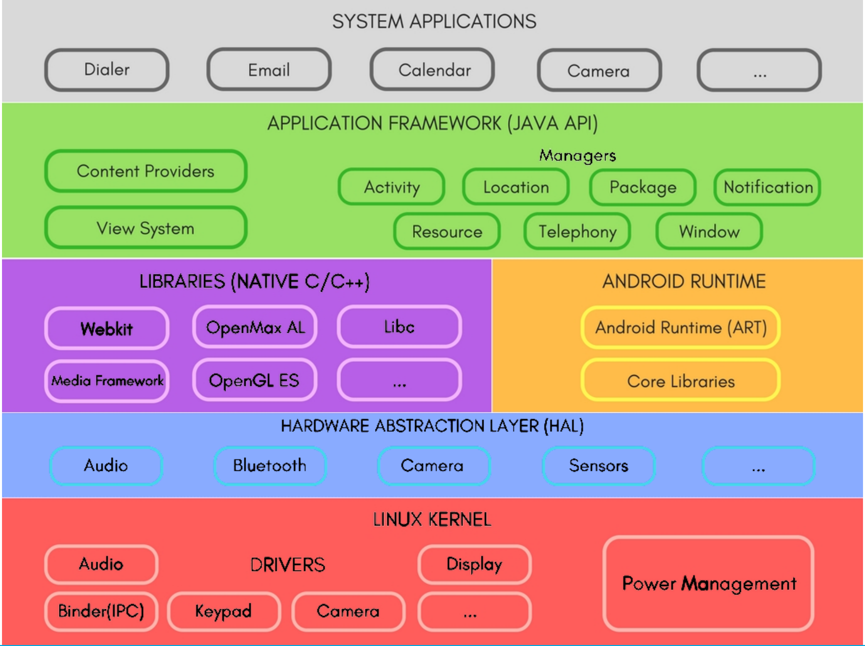
- `SELECT count(*) FROM data1 WHERE content MATCH 'linux'; /* 0.03 seconds */`
- `SELECT count(*) FROM data2 WHERE content LIKE '%linux%'; /* 22.5 seconds */`

- Mapping via rowid/docid (no primary key).
- Different query syntax
  - `"TABLE_NAME MATCH "+query`
  - Support for wildcards, proximity queries etc.
  - Boolean queries.

Android Java Services

- There are lots of services implemented in Java in Android.
- They abstract most of the native features to make them available in a consistent way.
- Accesible via `Context.getSystemService()`
- Read the docs for more.

Android's architecture – new



ActivityManager

- ActivityManager manages everything related to Android applications
  - starts activities,
  - manages their life-cycle,
  - fetches content from content providers,
  - handles non-responding applications.

TelephonyManager

- TelephonyManager provides access to information about the telephony services on the device.
- `TelephomyManager.EXTRA_STATE`
  - `EXTRA_STATE_RINGING`
  - `EXTRA_INCOMING_NUMBER`
- SIM info
- Location of the cell
- Operator info

PackageManager

- PackageManager is for manipulating already installed packages
  - get list of packages,
  - get/set permissions,
  - get details/resources,
  - install/uninstall package,
  - check that the application can handle intent of given type.

AlarmManager

- AlarmManager abstracts timers.
- Allows to set one time or repetitive timer.
- When a timer expires, the AlarmManager grabs a wakelock, sends an Intent to the corresponding application and releases the wakelock once the Intent has been handled.

ConnectivityManager

- ConnectivityManager manages network connections.
  - Falls back to other connections when one fails.
  - Notifies the system when one becomes available/unavailable.
  - Allows the applications to retrieve various information about connectivity.

WifiManager

- WifiManager provides an API to manage all aspects of WiFi networks
    - list, modify or delete configured networks,
    - get information about current WiFi network,
    - list currently available WiFis,
    - send intents on changes.

NotificationManager

- NotificationManager notifies the user of events that happen. This is how you tell the user that something has happened in the background.
- Notifications can take different forms:
    - A persistent icon that goes in the status bar and is accessible through the launcher,
    - Turning on or flashing LEDs on the device,
    - Alerting the user by flashing the backlight, playing a sound, or vibrating.

Other managers

- PowerManager
- LocationManager
- ClipboardManager
- ...

System server

- Starts all system services and managers.
- Runs services as:
  - Lights Service,
  - Vibrator Service,
  - Sensor Service,
  - ...

References 1/3

- Vogella: Getting started with Android development
  - `http://www.vogella.com/articles/Android/article.html`
- Mkyong: Android Tutorial
  - `http://www.mkyong.com/tutorials/android-tutorial/`
- Tutorials Point: Android Tutorial
  - `https://www.tutorialspoint.com/android/index.htm`
- Learn Android Development
  - `https://www.studytonight.com/android/`
- Documentation
  - `https://developer.android.com`

- Others
  - https://www.businessinsider.com/how-android-was-created-2015-3
  - https://www.digitaltrends.com/mobile/android-version-history/
  - https://www.techuntold.com/sequence-behind-android-os-names/
  - http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture
  - https://www.boldare.com/blog/differences-between-class-and-dex-files-in-java-android/
  - http://darutk-oboegaki.blogspot.cz/2011/03/usage-of-dx-dex-dx-dex.html
  - http://stackoverflow.com/questions/9593527/what-are-odex-files-in-android
  - http://www.addictivetips.com/mobile/what-is-odex-and-deodex-in-android-complete-guide/
  - https://web.archive.org/web/20171108061932/http://www.limbaniandroid.com/2014/04/how-to-display-datepickerdialog-in.html
  - https://stackoverflow.com/questions/45373007/progressdialog-is-deprecated-what-is-the-alternate-one-to-use
  - https://lief.quarkslab.com/doc/latest/tutorials/10_android_formats.html

References 3/3

- Others
  - http://thetechnocafe.com/how-to-create-widget-for-your-android-app/
  - https://www.slideshare.net/opersys/understanding-the-android-system-server
  - https://stackoverflow.com/questions/62138507/intentservice-is-deprecated-how-do-i-replace-it-with-jobinter
  - https://medium.com/mindorks/android-jobintentservice-for-background-task-e77bfa21fffa
  - https://android-developers.googleblog.com/2020/02/android-studio-36.html

Thank you for your attention!