1 Introduction

A) An Android application consists out of the following parts:

- **Activity:** A screen in the Android application
- **Services:** Background activities without UI
- **Content Provider**: provides data to applications, Android contains a **SQLLite DB** which can serve as data provider
- **Broadcast Receiver**: receives system messages, can be used to react to changed conditions in the system
- **Intent:**
    - allows the application to request and / or provide services. For example, the application call ask via an intent for another contact application.
    - Applications register themselves via - what is called - an **IntentFilter**.
    - Intends are a powerful concept as they allow to create loosely coupled applications.

B) **AndroidManifest.xml:** Any Android application is described in the file "AndroidManifest.xml". This file contains:

- all activities and
- all required permissions for the application; e.g., if the application requires network access it must be specified here.

C) **Activities** and **Layouts**

Layouts define how the graphical user interface for Activities appears to the users. The layout defines the UI elements, their properties and their arrangement. A layout can be defined via XML and via code at runtime. The XML way is usually preferred for a fixed layout while defining the layout via code is more flexible. You can also mix both approaches.

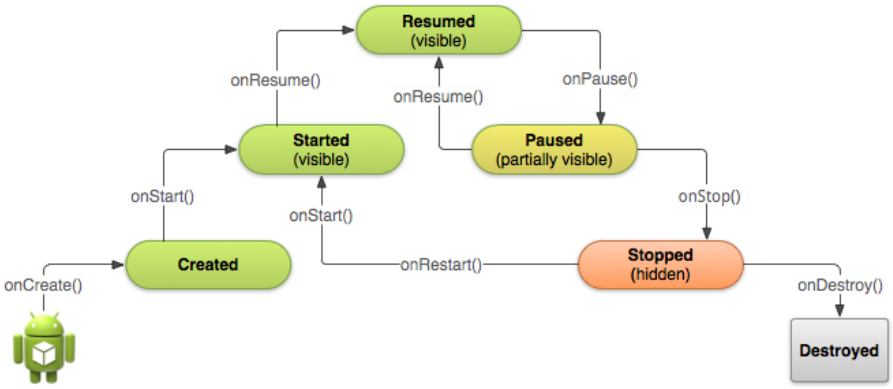
1.1 Activities and Lifecyle

The operating system controls the lifecycle of your application. At any time, the Android system may stop or destroy your application, e.g. because of an incoming call. The Android system defines a life cycle for activities via pre-defined methods. The most important methods are:

- onSaveInstanceState() - called if the activity is stopped. Used to save data so that the activity can restore its states if re-started

- onPause() - always called if the Activity ends, can be used to release resource or save data

- onResume() - called if the Activity is re-started, can be used to initiate fields



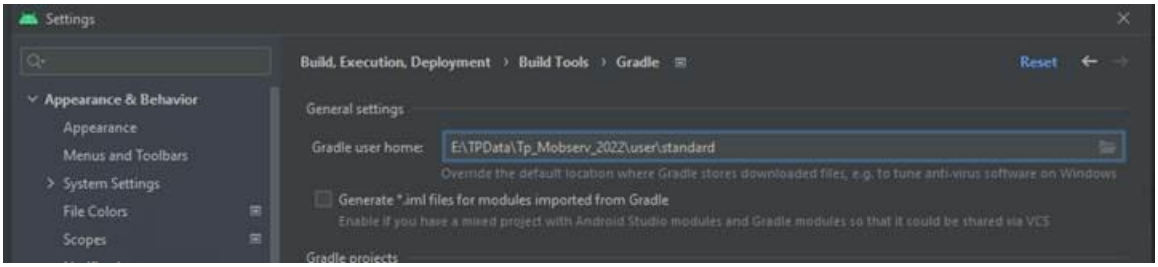
You are encouraged to test the following two examples:

- http://developer.android.com/training/basics/activity-lifecycle/index.html
- http://www.vogella.com/tutorials/AndroidLifeCycle/article.html

2 HelloWorld [30 minutes]

- Go to **E:\TpData\Tp_Mobserv_2022**
- Double click on "start_android_lab.bat" to run the script file which will create all the necessary paths for our lab session.
- Install Type: **Custom to set the** SDK, you may need to fix the path manually.
  - Select custom setting or just **Configure/settings**
  - Click on "edit" to (re)set the Android SDK location to: "`E:\TpData\Tp_Mobserv_2022\Sdk`"
- **Start a new Android Studio project**, see http://developer.android.com/sdk/installing/create-project.html
  - Activity: **Empty activity**
  - App name: HelloWorld
  - Package Name: fr.eurecom
  - Project language: Java or Kotlin
  - **Save location**: "`E:\TPData\Tp_Mobserv_2022\user\%USER%\Workspace\HelloWorld`

" where %USER% must be your user name (this is already created when you run "start_android_lab.bat" as a first step).
- Target: phones and tablets with **min SDK API 2x** (this is just a suggestion), click on "**Help me choose"**
- Finish



NOTE: Android AVD location is set to : E:\TPData\Tp_Mobserv_2022\src\avd.

Create an AVD:

- Phone → Pixel 5→ R (API level 30, x86 image). Finish
- Action : play

NOTE: In case your initial java code prompts for errors, you will have to *fix* two issues regarding *Gradle* and *Style.xml* by following the 2 points below. You will need to repeat this "fix" every time you create a new empty project:

- Add the needed maven libraries for you project:
  - Go to "Gradle Scripts" (on the left panel space)
  - Chose "build.gradle (Project: HelloWorld)".
  - Then, make the following additions (in green) on both repository trees:
  - Then, go to menu → 'Build' → 'Rebuild project'

```
repositories {
    jcenter()
    maven{
        url "https://maven.google.com"
    }
}
```

- Fix the "styles.xml" issue:
  - Select from the left pane: app → res → values → styles.xml
  - Modify the following line (in yellow) as below:

```
<style name="AppTheme" parent="Base.Theme.AppCompat.Light.DarkActionBar">
```

- When promoted for Firewall, just Cancel.
- Run the application by clicking the play button.
  - Before running, check if there is a running "adb" process. If so, **kill** the existing **adb** process:

- Open the windows task manager (ctrl+shift+esc)
- Under the processes tab, find "image name" **adb**
- Right click on it and select the option "End Process Tree" from the popup menu

- **Launch emulator** and ... *be patient*.
- Run → debug 'app' → Select Deployment Target form pops up
  - You chose to click on the checkbox at bottom-left of the form to remember your AVD section
  - *Select your AVD* and click on 'ok'
  - Be patient; it may take some time to install the APKs to the AVD
  - **NOTE**: If you have connected a **real device** with a usb cable, please refer to the instructions here: https://developer.android.com/training/basics/firstapp/running-app.html

- You should see "HelloWorld" in a `textview` component in the middle of your application frame.

- Change "HelloWorld" to "HelloMoon" from app > res > layout > activity_main.xml, and check the results on the emulator.
  - Change the color, font, position etc. and run again.
  - Check the logcat

3 Implicit Intents (90 minutes)

Android supports
- explicit intents and
- implicit intents.

An explicit intent name the component it will invoke, whereas an implicit intent asks the system to perform a service without telling the system which component (piece of code, i.e. a java class in the case of java) should do this service.

In an implicit Intent you specify
- Action: e.g., view
- URI: e.g., webpage

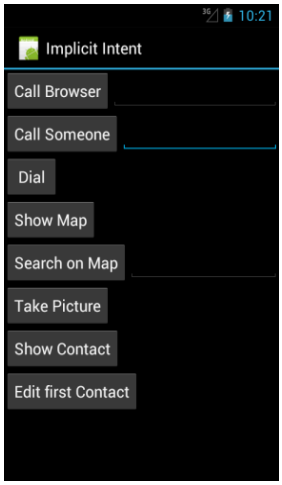
Moreover, the system will find an application that is registered for this event, e.g. a browser.

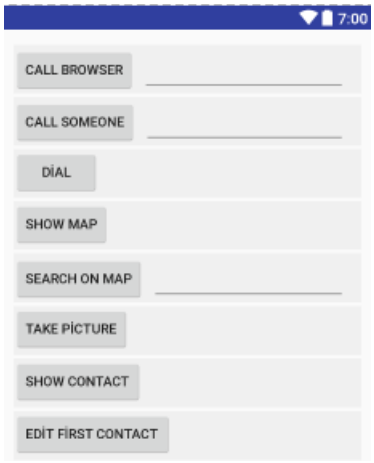
Useful methods:
- **`startActivity(Intent)`:** if you do not need a return value from the called activity
- **`startActivityForResult()`** and **`onActivityResult()`**

An intent can contain data when it requires results through the following methods:
- `putExtra()` if you want to add data to the intent
- `getAction()`, `getData()` and `getExtras()` if you want to receive the data and url from this Intent.

3.1 Create a Project

1. Application name: Implicit Intent
2. Package name: fr.eurecom.implicitintent
3. Activity type: Empty Activity
4. Create Activity: MainActivity





3.2 Edit Layout

1. Add 8 Buttons: While being in the 'Design' tab (not the 'text' tab) at the bottom of the 'activity_main.xml', chose from the 'palette' on your left a 'button' and drag-n-drop it on the design frame (on the right).
   - Call browser
   - Call someone
   - Dial
   - Show map, position of Eurecom
   - Search map, e.g. valbonne
   - Take a picture
   - Show contact
   - Edit the first contact
2. For every button on the design frame view, click `properties' (the pane on the right side of the design view)→ view all properties and set the following properties
   - Set the following
     - `id` - they have to be unique, e.g. "button2"
     - `text` - the text written on the buttons
     - `onClick`: this is a function/handler - the function to be called when button is clicked, e.g. "callIntent"

       NOTE: Whatever you changed above, is mapped to an entry in the text view along the lines of "android:xxx", e.g. `'android:onClick="callIntent"'`
   - Keep the `android:onClick` handler the same for all the buttons, and name it "callIntent"
3. Add three EditText for calling a browser, call someone, and search a map:
   - Chose from palette "`PlainText`". That is actually appearing in the text view frame as "`EditText`" in the xml code.

NOTE: If you click on the "text" (not the "design"), you will see that the items (buttons,textviews,edittexts) are placed based on absolute coordinates and sizes. This is because by default, the "empty activity" uses this layout style: "ConstraintLayout". If you wish to create GUIs that automatically adjust to other screen sizes, you need to dig into the details of other available layouts, e.g. pallet→ RelativeLayout or GridLayout, etc..

3.3 Add uses features and permission to the manifest.xml file

1. Add in the AndroidManifest.xml file (inside manifest, before application tag starts) that your application will use features as follows:

```
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.location.gps" />
```

You need to state this App will use camera. Since for newer version of android photos embed GPS coordinates to pictures files, you have to mention you will need GPS features.

2. Open the AndroidManifest.xml file, and add the following permissions:

- CALL_ CALL_PRIVILAGE
- CALL_PHONE
- INTERNET
- CAMERA
- READ_CONTACTS
- WRITE_EXTERNAL_STORAGE
- READ_EXTERNAL_STORAGE
- ACCESS_FINE_LOCATION

Before the application tag, add for each type of a privilege listed above a "uses-permission" tag as follows:

<uses-permission android:name="android.permission.READ_CONTACTS " />

3.4 Implement the onclick function for all buttons

Implementation of the first button is given below, you need to implement the remaining EditText and buttons.

NOTE: "R.*" traditionally denotes the "**R"** resources of the application.

```
public void callIntent(View view) {
       Intent intent = null;
       switch (view.getId()) {
         case R.id.button1:
             EditText textBrow = (EditText) findViewById(R.id.edit_url);
             String url = textBrow.getText().toString().trim();//trim to avoid
                                                    // possible white spaces
             intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://" + url));
             startActivity(intent);
             break;
         case R.id.button2:
              //…
}
```

- You need the following actions (see developer.android.com/reference/android/content/Intent.html):

- ACTION_VIEW, ACTION_CALL, ACTION_DIAL, ACTION_EDIT, ACTION_IMAGE_CAPTURE, ACTION_INSERT, ACTION_PICK

- For the URI, see http://developer.android.com/training/basics/intents/sending.html
- For the camera, the development is different than the others. Follow this link to figure out how to take a picture with a camera intent: https://developer.android.com/training/camera/photobasics#java
Unlike the other intents, after the user takes a picture, the camera intent returns with a result (in this case result is a Uri that is a picture). **OPTIONAL**: For extended practice, you can try to save this picture in a directory inside the phone. To do so, you might require an SD card inserted to your emulator (which should be by default there). The link above also provides code samples to use the resulting image to convert it into BitMap variable or save to the storage.

NOTE: The emulator might not have an app called "MyFiles" in order to verify the image files are created properly. But on a real device the pictures should be properly created.

Note: You also need to **check** and **request** a **runtime permission** form the user for starting certain activities, such as calling, taking a picture, and writing in calendar.

```
private static String[] PERMISSIONS_STORAGE = {
        Manifest.permission.READ_EXTERNAL_STORAGE,
        Manifest.permission.WRITE_EXTERNAL_STORAGE,
};
public static void verifyStoragePermissions(Activity activity) {
    // Check if we have write permission
    int permission = ActivityCompat.checkSelfPermission(activity,
Manifest.permission.WRITE_EXTERNAL_STORAGE);
    if (permission != PackageManager.PERMISSION_GRANTED) {
        // We don't have permission so prompt the user
        ActivityCompat.requestPermissions(
                activity,
                PERMISSIONS_STORAGE,
                1
        );
    }
}
// Request runtime camera permission.
if (ActivityCompat.checkSelfPermission(this, Manifest.permission.CAMERA) !=
PackageManager.PERMISSION_GRANTED) {
                    ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.CAMERA}, 2);
                }
```

The code block above will check if this activity has permissions to Storage, if not it will show a prompt to ask for it from the user as follows:
NOTE: Once the permission is given, on next debugs the permission for will still be there, so you might need to uninstall the app from the emulator before debugging again to test it once again. (Permissions can also be given from the Settings → Apps → ImplicitIntent page

For more information, have a look at
https://developer.android.com/training/permissions/requesting.html

```
@Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == 1 && resultCode == RESULT_OK) {
            Bundle extras = data.getExtras();
            Bitmap imageBitmap = (Bitmap) extras.get("data");

            String filename="test" ;
            File outputFile =new
File(Environment.getExternalStorageDirectory(),"photo_"+filename+".png");
            try {
                FileOutputStream out =new FileOutputStream(outputFile);
                imageBitmap.compress(Bitmap.CompressFormat.PNG,100,out);
                out.flush();
                out.close();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
         super.onActivityResult(requestCode, resultCode, data);

    }
```

- You can override the onActivityResult to define what will happen to the image when intent returns the resulting Uri. Switch case is there in case you might have more than one intents in your app returning with a result.

Exercises

1. When and for what purpose are the following methods called during the lifecycle of an Activity?

   ➔ `onCreate(), onStart(), onResume(), onDestroy(), onStop(), onPause().`

2. In the ImplicitIntent example, could we use a different OnClick function per button?
3. Use an example to illustrate the relationship between different application components.
4. Show an app chooser based on the instructions online here:
   http://developer.android.com/training/basics/intents/sending.html#AppChooser
5. Add another button that, when pressed, adds events to the calendar. Consider the following:
   a. Use ACTION_INSERT to create the intent, and
      `intent.setData(CalendarContract.Events.CONTENT_URI);`
   b. Additional link: http://www.vogella.com/articles/AndroidCalendar/article.html

4 References

[1] http://developer.android.com/guide/topics/fundamentals.html
[2] http://www.vogella.de/android.html
[3] http://developer.android.com/resources/index.html
[4] http://code.google.com/p/android-for-gods/w/list