Real-Time Embedded Systems

DT8025, Fall 2016
http://goo.gl/AZfc9l

Lecture 8

Masoumeh Taromirad
m.taromirad@hh.se

Center for Research on Embedded Systems
School of Information Technology

Smart phones actors

Google

- Platform: Android
- Language: Java
- Development: Eclipse

Apple

- Platform: iOS
- Language: Objective C
- Development: Xcode

Microsoft

- Platform: Windows Phone
- Language: C#
- Development: Visual Studio

We choose Android because it is more open, it is easier to distribute apps and most of you are familiar with Java and Eclipse. Java libraries are available. Swing is not available, UIs are done in a different way.

A paradigm shift

Before

Large teams of programmers involved in large pieces of software following a software engineering process.

Now

One (or a few) programmers developing apps that do very specific things and make use of other apps for standard things.

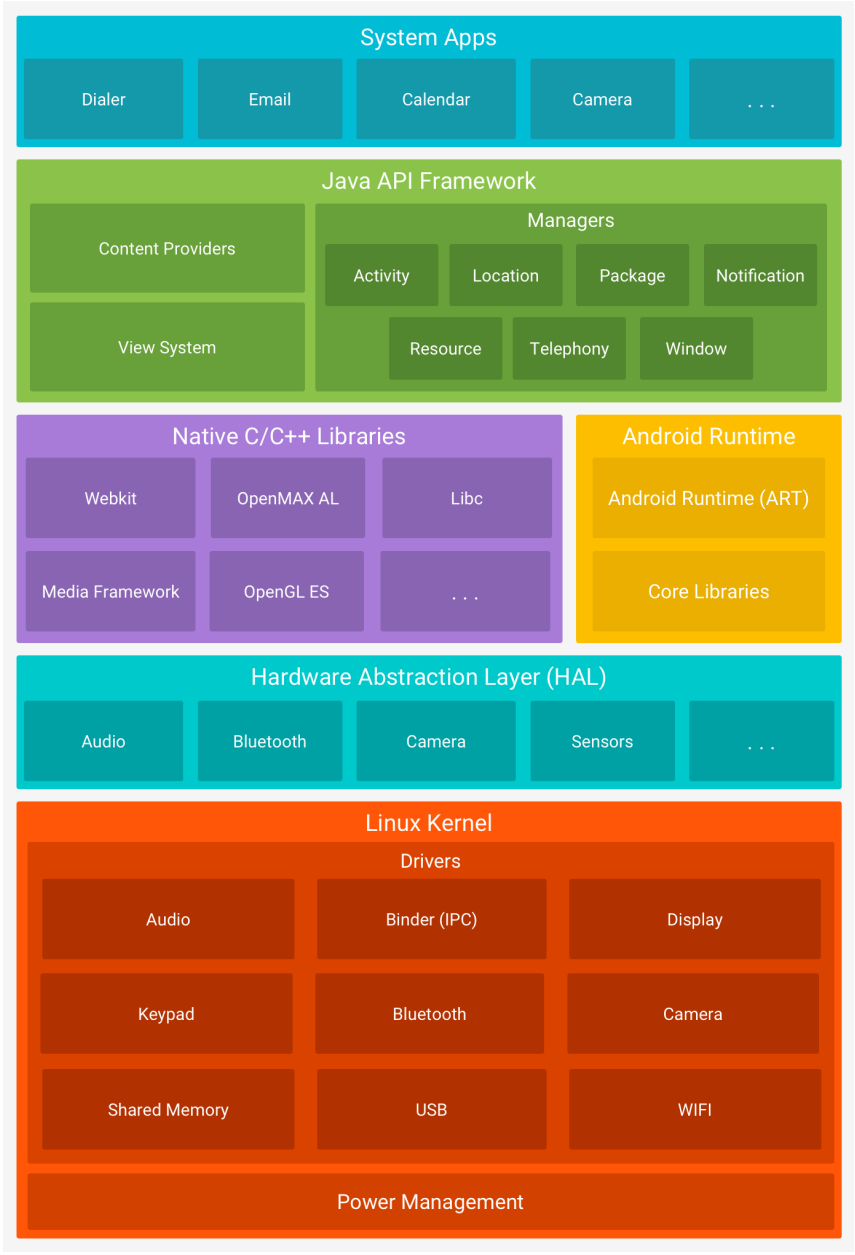
Before

Big releases, including distributing to customers or retailers.

Now

Just distribute online or use some app market.

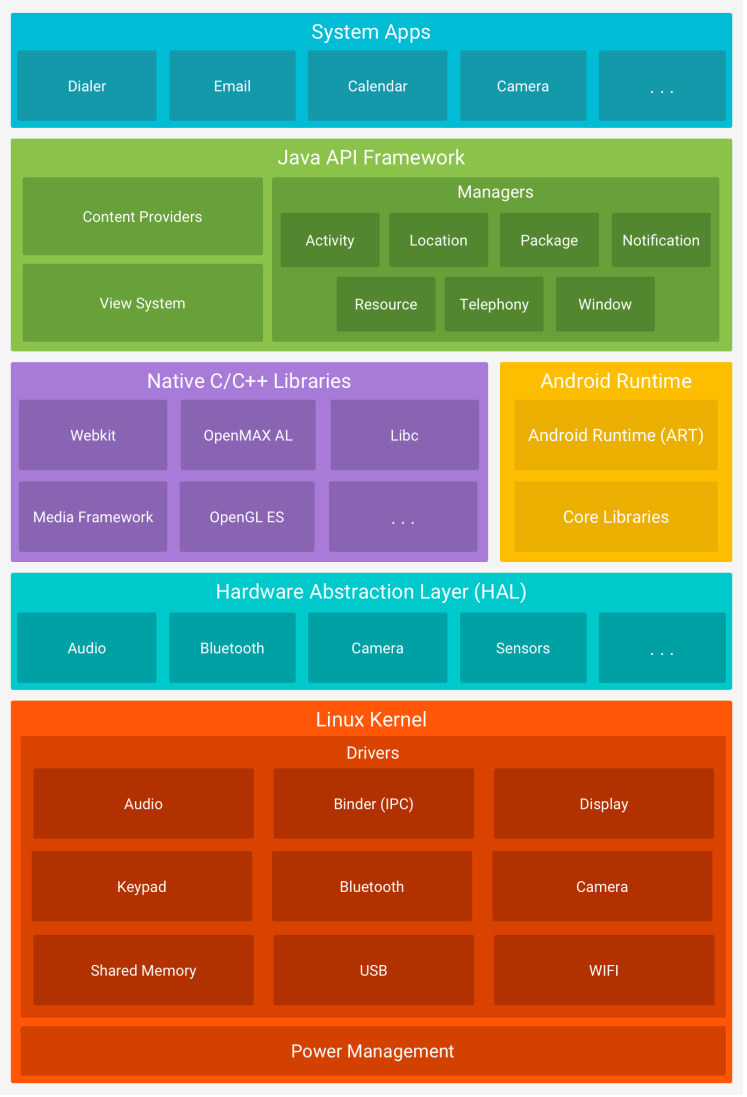
The Platform Architecture



Linux kernel

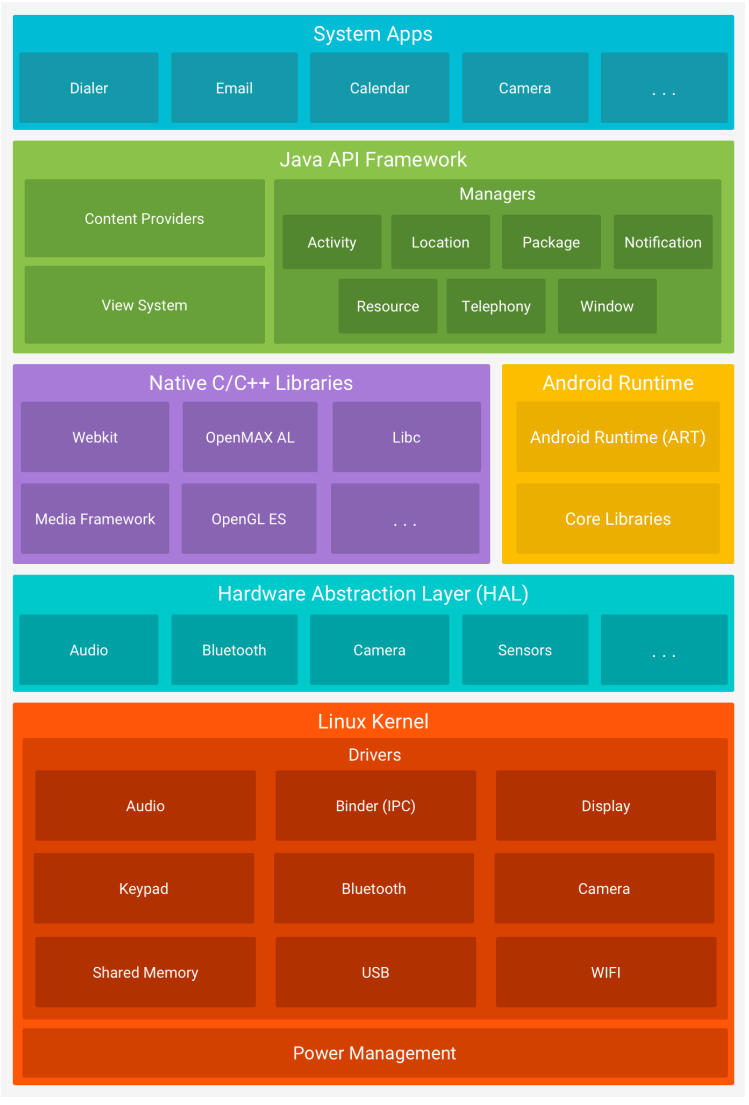
- Device drivers
- Process management (process creation, memory management)
- Interprocess communication

We will not use this directly but it is important to know that each application that is started is a Linux user! So: applications run in isolation from other apps!



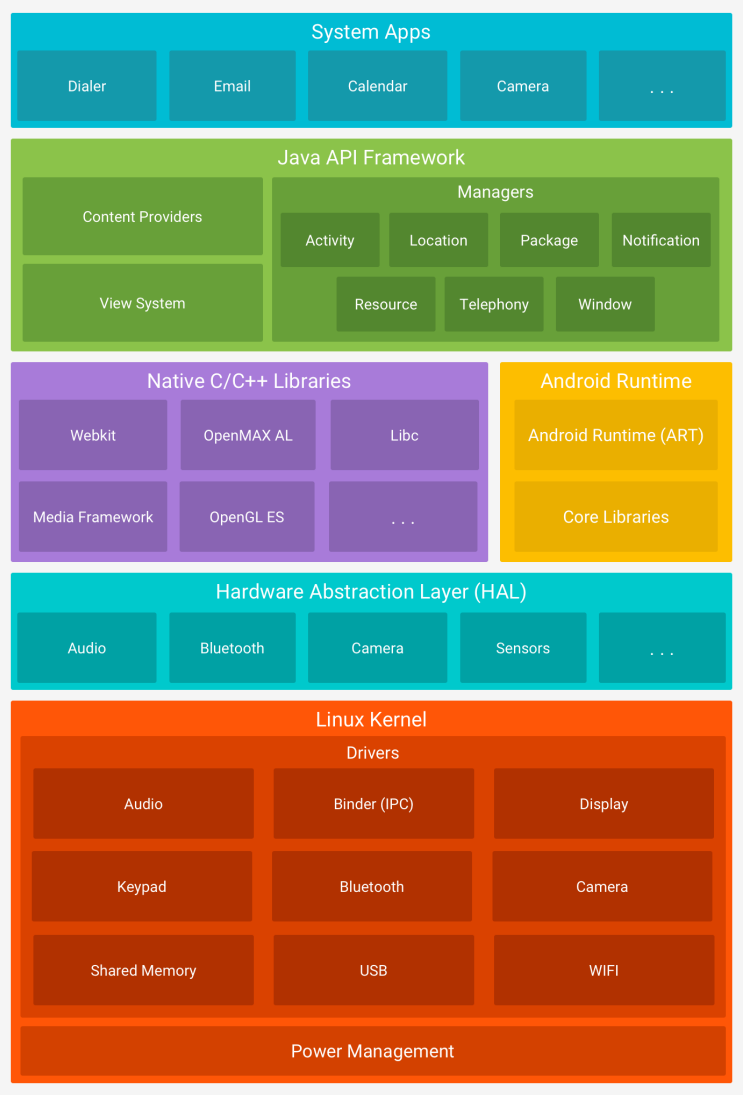
Hardware Abstraction Layer (HAL)

- standard interfaces that expose device hardware capabilities to the higher-level.
- to access device hardware, the Android system loads the library module for that hardware component.



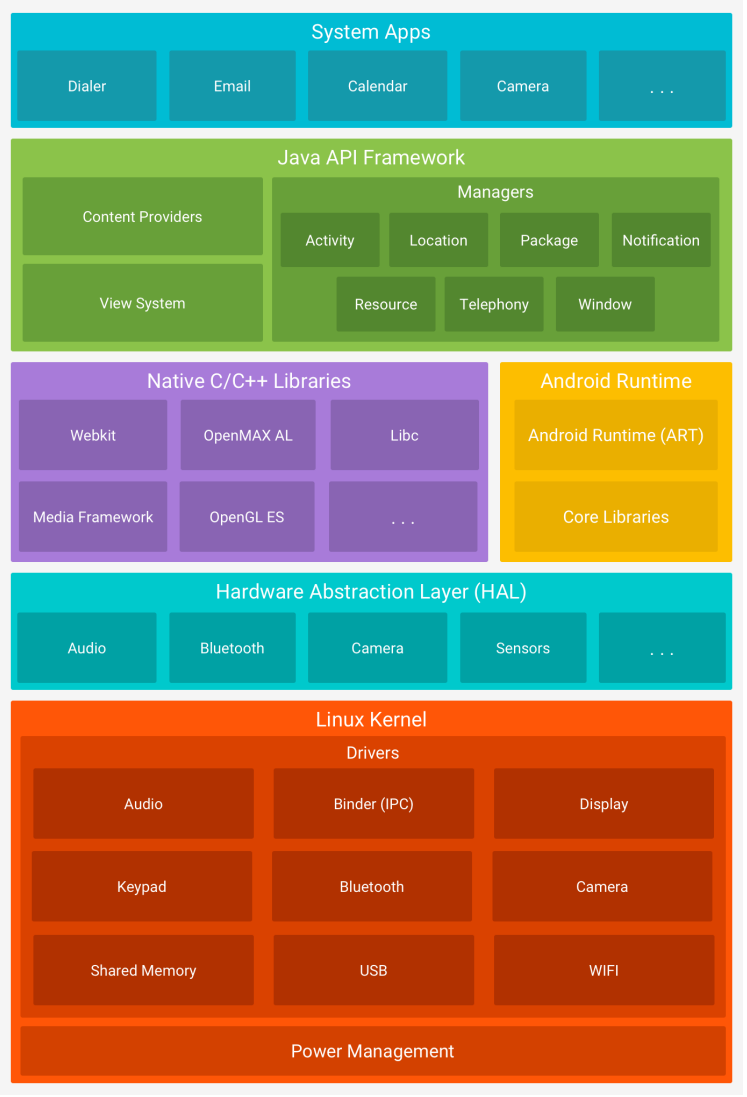
Android Runtime (ART)

- Every Android application runs in its own process, with its own instance of the Android Runtime (ART).
- ART has been written so that a device can run multiple VMs on low-memory devices efficiently.



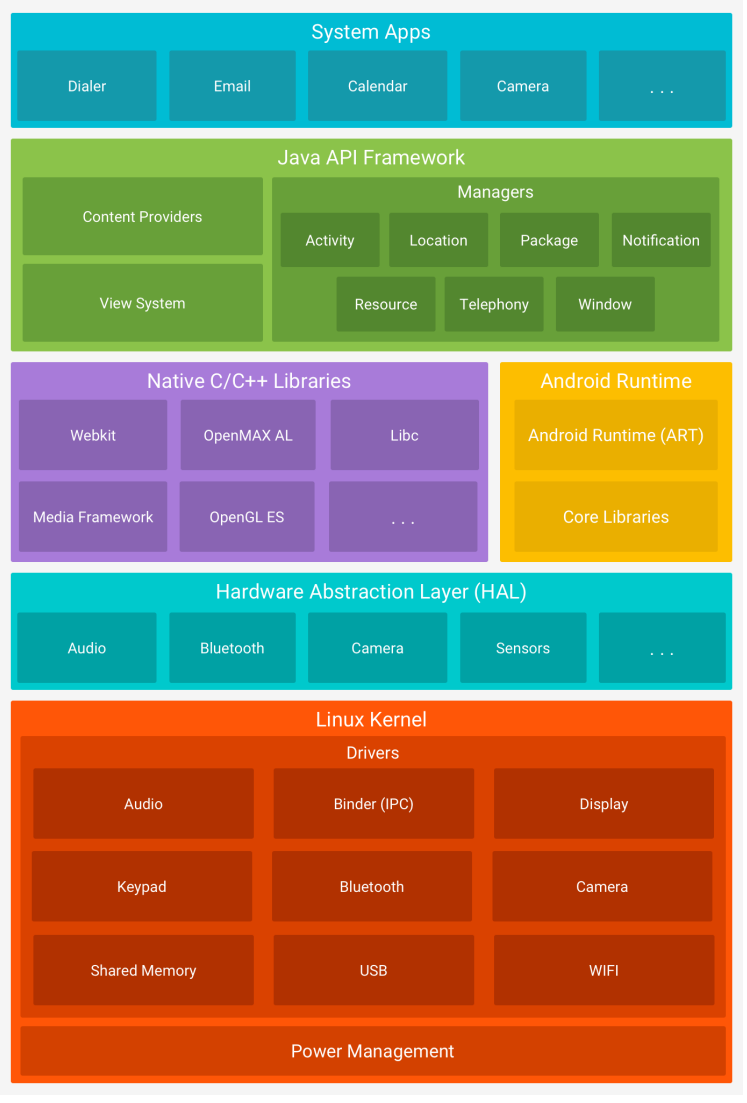
Android Runtime (ART)

- The ART VM executes DEX files, a bytecode format which is optimized for minimal memory footprint.
- The ART relies on the Linux kernel for underlying functionality such as threading and low-level memory management.



Native C/C++ Libraries

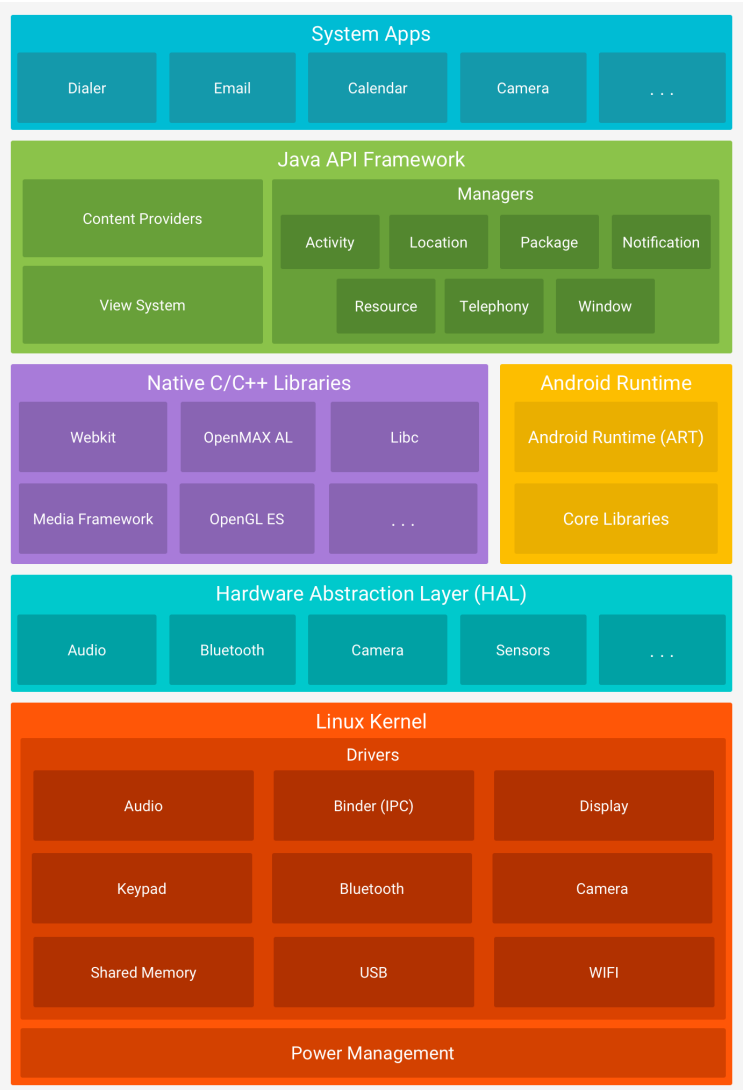
Many core Android system components and services, such as ART and HAL, are built from native code that require native libraries written in C and C++.



Java API Framework

- The entire feature-set of the Android OS is available to you through APIs written in the Java language.

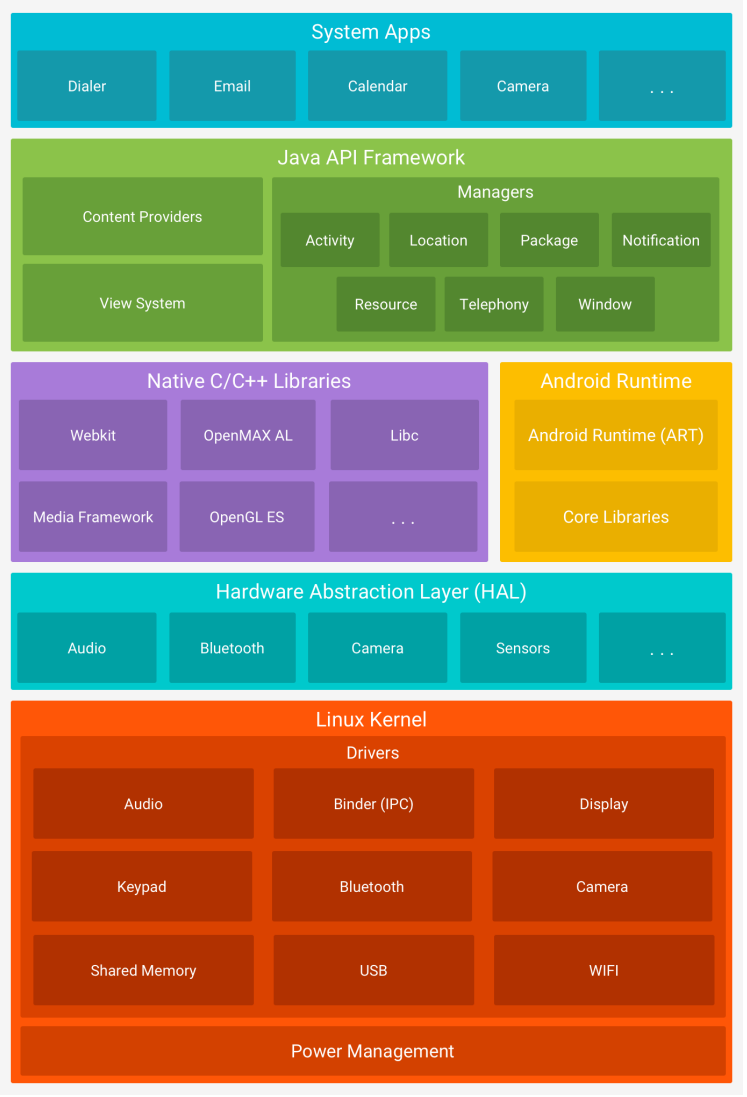
View System, Resource Manager, Notification Manager, Activity Manager, Content Providers, ...



System Apps

- Android applications
- Android comes with a set of core apps for email, messaging, calendars, internet browsing, contacts, and ....

Apps included with the platform have no special status among the apps the user chooses to install.



Hello World!

What could an Android *hello world* application be like?

Android 101

Download and install Android SDK (ADT Bundle) from:
`http://developer.android.com/sdk/`

Android 101

In order to run your Hello World! app:

1. connect your Android cell-phone and enable USB debug on your device, or
2. install a virtual device.

Hello World!

1. Create a new Android Virtual Device (AVD)
2. Create a new Android application project
3. Run and see the behavior
4. Check out the res/layout and res/values

What did we see?

A screen: this will be reflected in the program as an Activity.

UI components: in the program these are Views.

Applications

The AndroidManifest puts together

Activities

Services

There is no main!

BroadcastReceivers

ResourceProviders

Intents

to build an application.

Layouts

Drawables

Values

Some code

```
void onClick(View view) {
  try{
    String address = addressfield.getText().toString();
    address = address.replace(' ', '+');
    Intent geoIntent
        = new Intent(android.content.Intent.ACTION_VIEW,
                     Uri.parse("geo:0,0?q=" + address));
    startActivity(geoIntent);
  } catch (Exception e)
}
```

This is something a button can do when clicked! We just have to associate this function with the right button!

Some code

```
void onClick(View view){
  try{
    String address = addressfield.getText().toString();
    address = address.replace(' ', '+');
    Intent geoIntent
        = new Intent(android.content.Intent.ACTION_VIEW,
                     Uri.parse("geo:0,0?q=" + address));
    startActivity(geoIntent);
  } catch (Exception e)
}
```

A kind of `View` called an `EditText` can be used this way!

Some code

```
void onClick(View view){
  try{
    String address = addressfield.getText().toString();
    address = address.replace(' ', '+');
    Intent geoIntent
       = new Intent(android.content.Intent.ACTION_VIEW,
                    Uri.parse("geo:0,0?q=" + address));
    startActivity(geoIntent);
  } catch (Exception e)
}
```

An `Intent` has an action and some data. It is something the program can ask the OS to deliver to some app that can do this!

Some code

```
void onClick(View view){
  try{
    String address = addressfield.getText().toString();
    address = address.replace(' ', '+');
    Intent geoIntent
      = new Intent(android.content.Intent.ACTION_VIEW,
                   Uri.parse("geo:0,0?q=" + address));
    startActivity(geoIntent);
  } catch (Exception e)
}
```

This is where the program asks the OS to deliver the `Intent`.

Activities

Each activity has a window to display its UI. It typically fills the screen.

An application usually consists of multiple activities that are loosely bound to each other.

Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time.

Activities

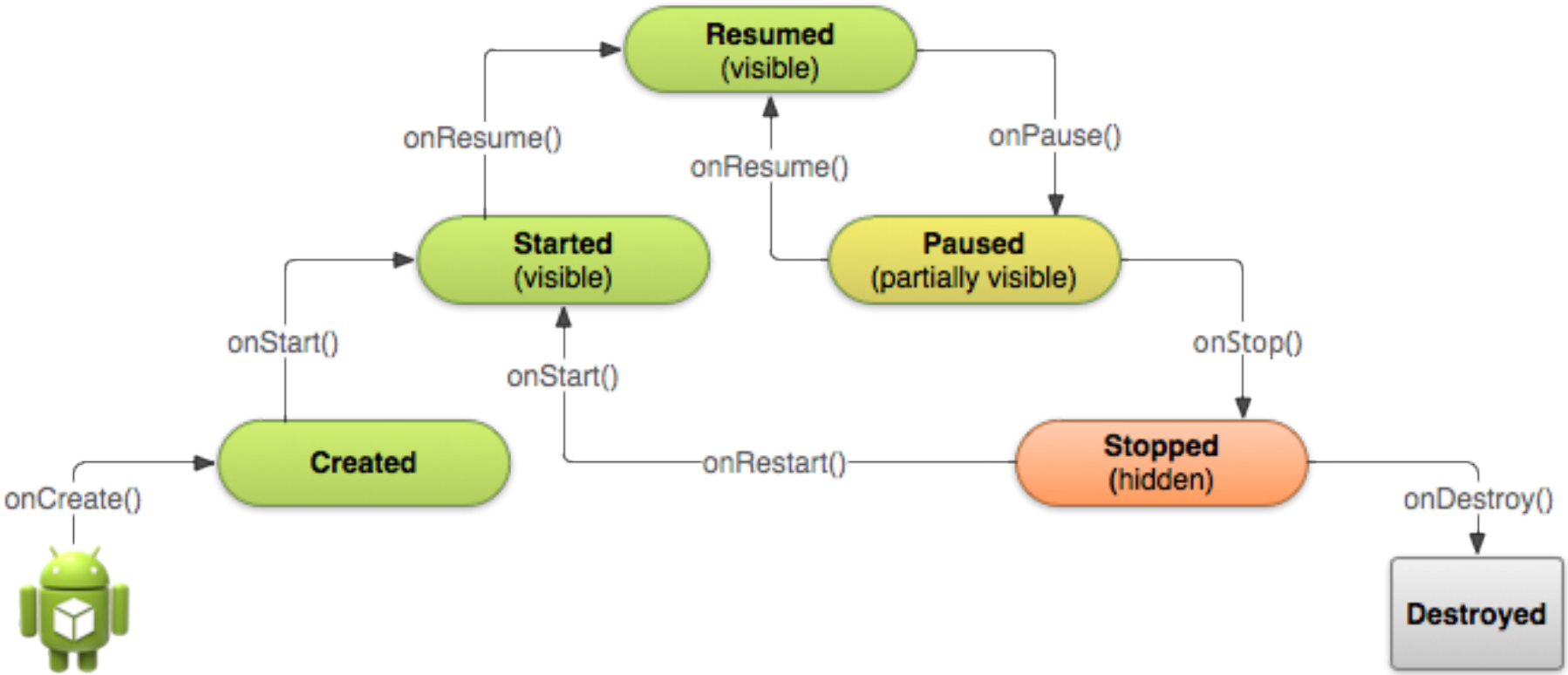
Activities can start each other, as a result:

- Started activity pushed on top of the stack, taking user focus,
- Starting activity stopped and remains in the stack.

The Skype app

- Main
- Contacts
- Profile
- Latest

Life cycle


Resumed
(visible)
onResume()
onResume()
onPause()
Started
(visible)
Paused
(partially visible)
onStart()
onStart()
onStop()
Created
onRestart()
Stopped
(hidden)
onCreate()
onDestroy()
Destroyed

An application with one activity

```
public class Quiter extends Activity{

  public void onCreate(Bundle savedInstanceState){
     super.onCreate(savedInstanceState);

     setContentView(R.layout.main);

     Button q = (Button)findViewById(R.id.quitButton);

     q.setOnClickListener(
           new Button.OnClickListener(){
              public void onClick(View view){
                 finish();
              }});
   }
}
```

The AndroidManifest (generated automatically!)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="...>
    <application android:label="@string/app_name"
                 android:icon="@drawable/icon">
            <activity android:name="Quiter"
                      android:label="@string/app_name">
                <intent-filter>
                    <action android:name=
                     "android.intent.action.MAIN" />
                    <category android:name=
                     "android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
    </application>
</manifest>
```

The resources: main layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="...
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <Button
      android:id="@+id/quitButton"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:text="@string/quitText"
      />
</LinearLayout>
```

The resources: string values

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Quitter</string>
    <string name="quitText">Stop this!</string>
</resources>
```

The resources: drawables

res/drawable-hdpi/icon.png

Exploring the life cycle

We implement all the methods that are called by the system:

```
public class Quitter extends Activity{
    public void onCreate(Bundle savedInstanceState){...}
    public void onPause(){...}
    public void onStop(){...}
    public void onDestroy(){...}
    public void onResume(){...}
    public void onStart(){...}
    public void onRestart(){...}
}
```

Don't forget to call `super.onSomething` before doing other stuff when you override these methods!

Exploring the life cycle

We can use `android.util.Log` to produce debugging messages in the development terminal.

```
public void onResume(){
  super.onResume();
  Log.d("quitter", "onResume");
}
public void onStart(){
  super.onStart();
  Log.d("quitter", "onStart");
}
public void onRestart(){
   super.onRestart();
   Log.d("quitter", "onRestart");
}
```