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.

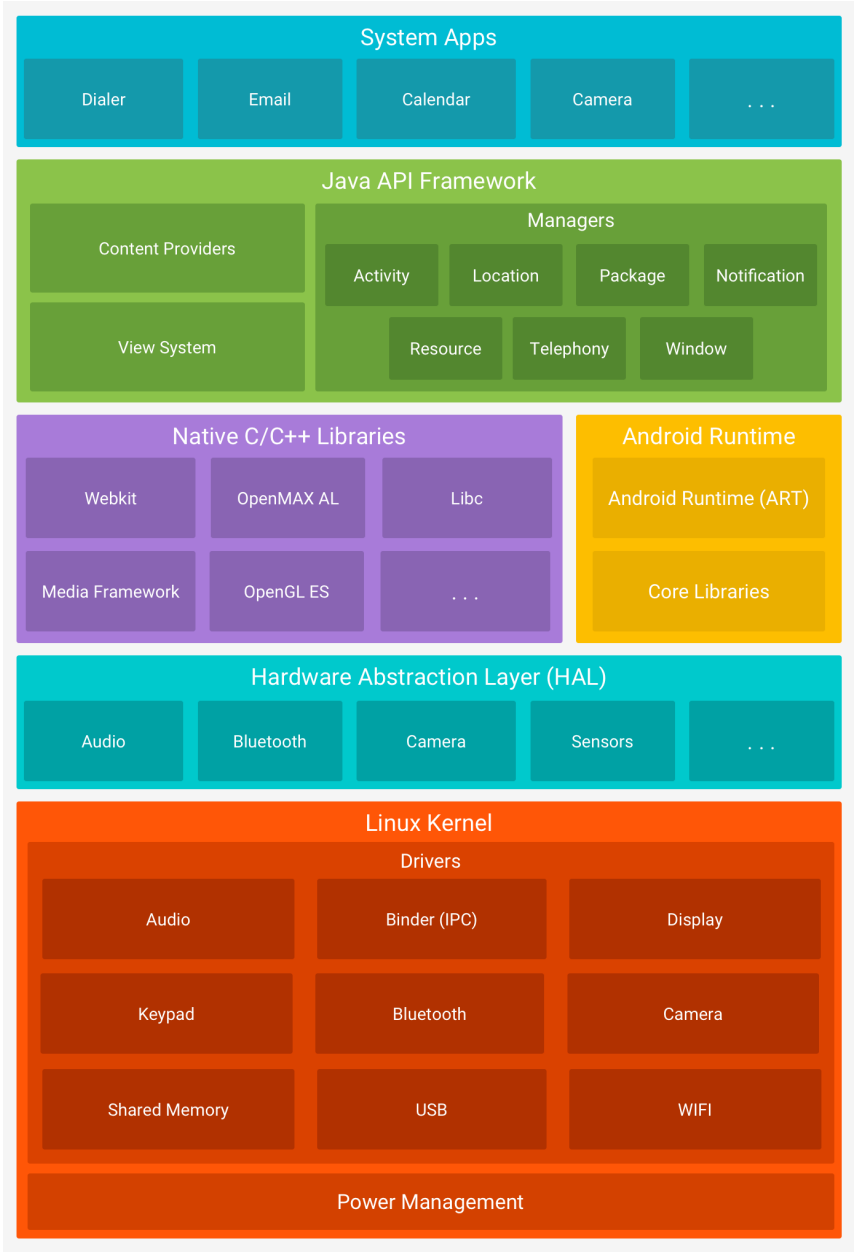
Before

Big releases, including distributing to customers or retailers.

Now

Just distribute online or use some app market.

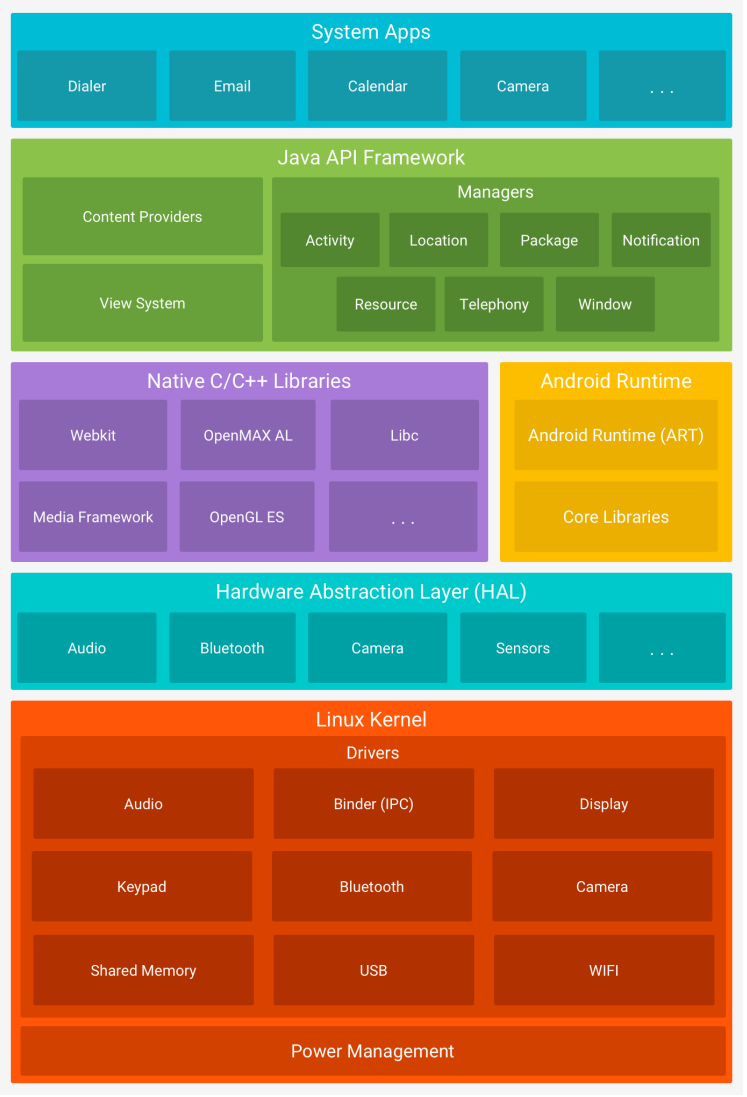
The Platform Architecture



Linux kernel

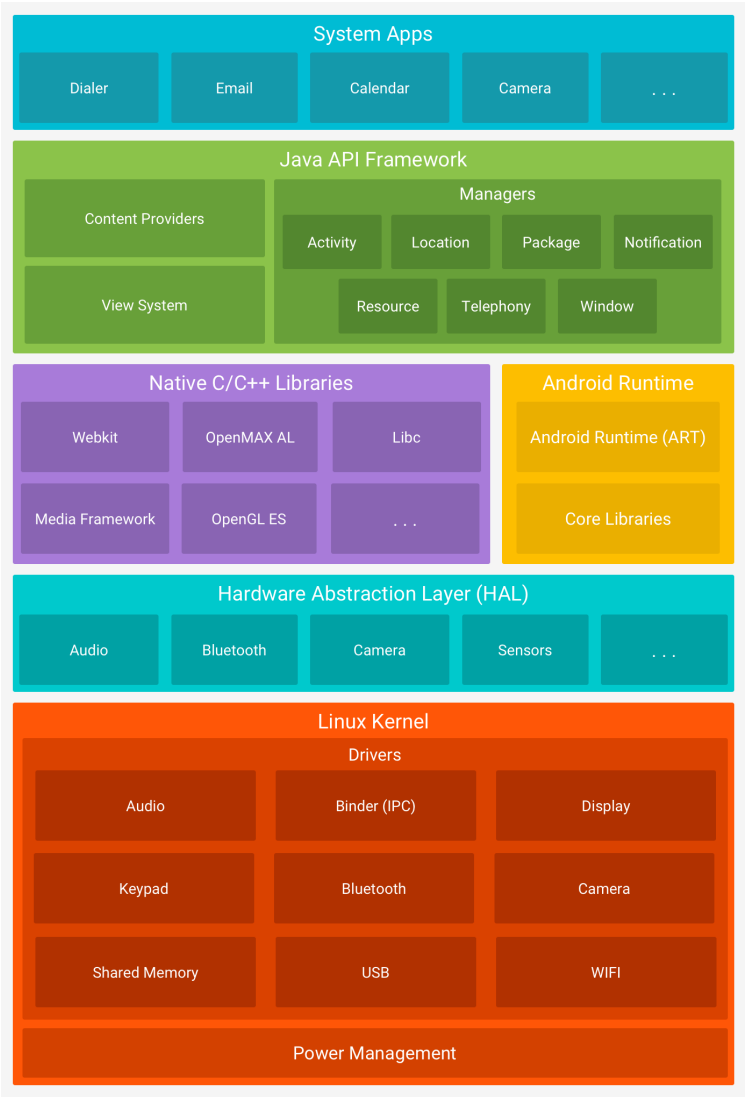
- Device drivers
- Process management (process creation, memory management)
- Interprocess communication

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!



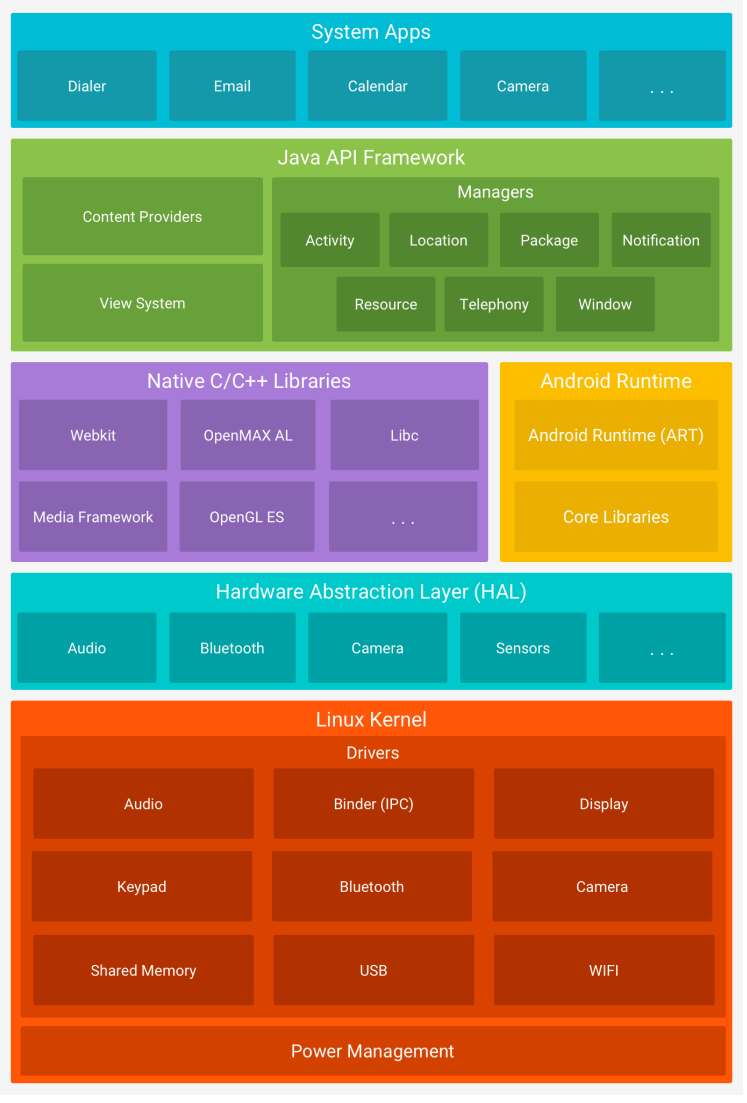
Hardware Abstraction Layer (HAL)

- 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.



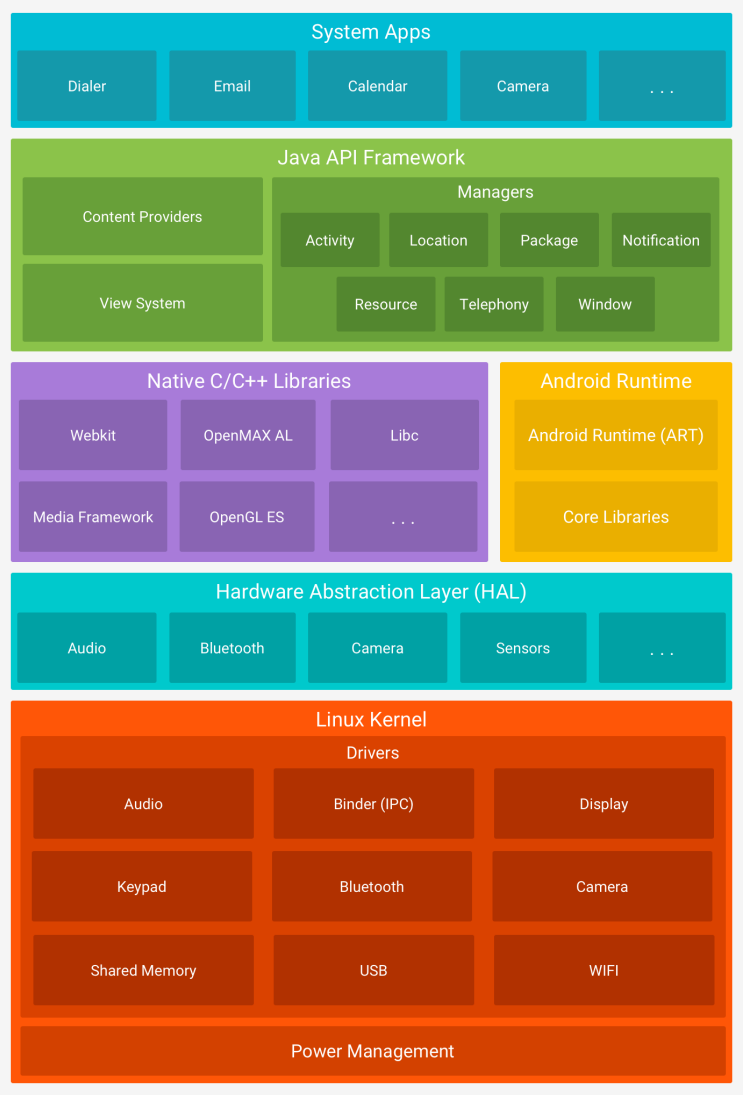
Android Runtime (ART)

- 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.



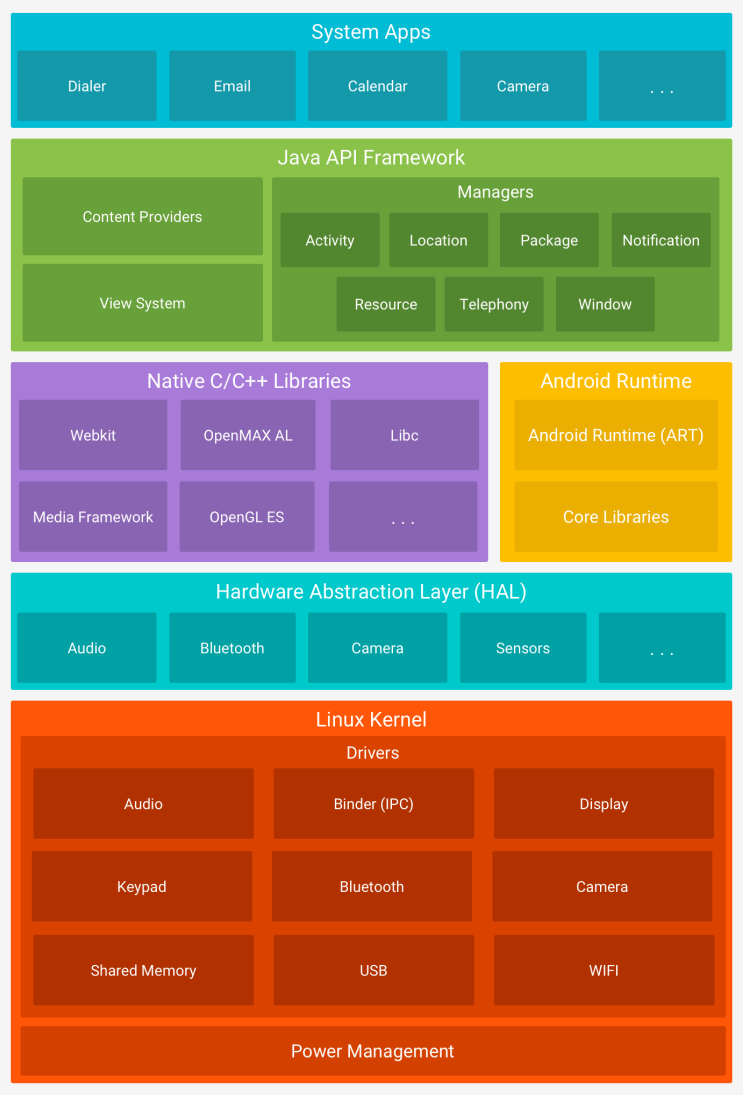
Android Runtime (ART)

- 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

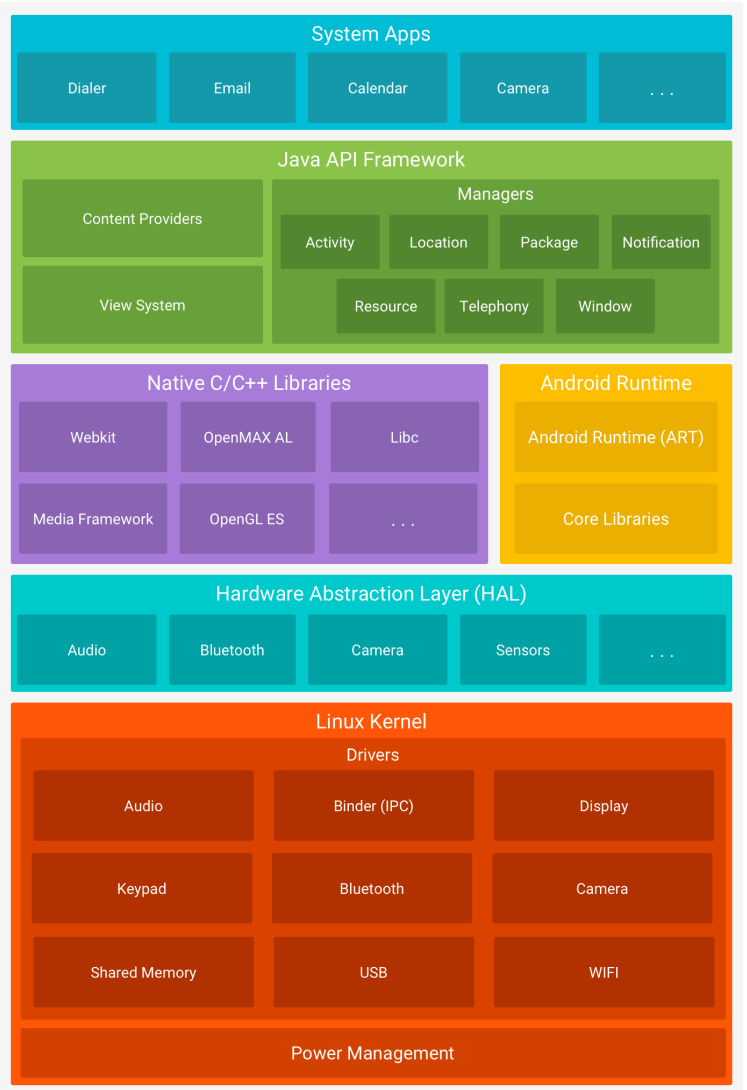
Many core Android system components and services, such as ART and HAL, are built from native code that require native libraries written in C and C++.



Java API Framework

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

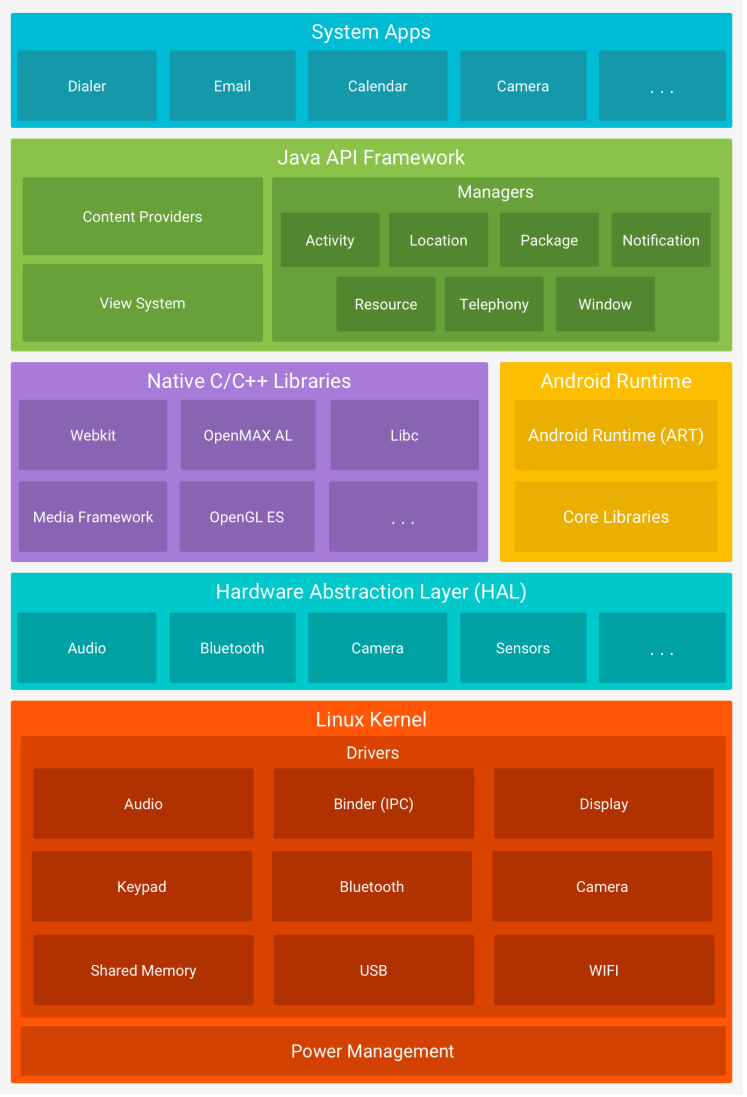
View System, Resource Manager, Notification Manager, Activity Manager, Content Providers, ...



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

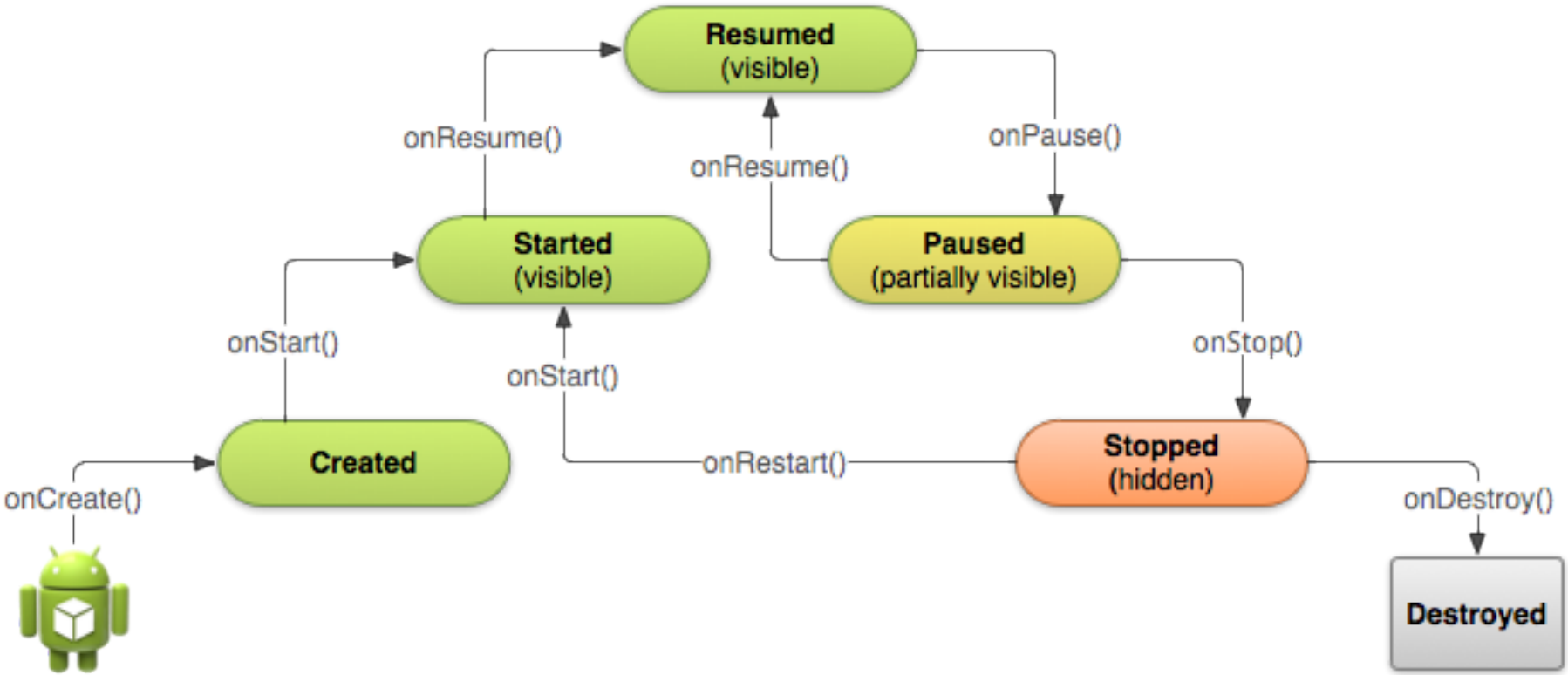
Activities can start each other, as a result:

- 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");
}
```

Processes in Android

Upon starting an application: a new Linux process with a single thread of execution.

By default, all components in the same process and thread (called the "main" thread).

Upon starting a component within an existing application: component is started within that process and uses the same thread of execution.

Keeping the UI reactive

Single thread model: challenge for reactivity.
Time consuming operations: separate threads ("background" or "worker" threads).

Note: the Android UI toolkit should not be accessed from outside the UI thread (not thread-safe UI methods)

The main thread

Run event listener and rest otherwise!

Posting *runnables*

To ask the UI thread to run some code:

```
public boolean post (Runnable action)
public boolean postDelayed (Runnable action,
                            long delayMillis)
```

Runnable

The interface `java.lang.Runnable` represents a command that can be executed.
A class that implements `Runnable` has to provide a method

```
public void run()
```

Threads in Java

Also used to start new threads in a Java program. This is how:

- Create a `Thread` passing a `Runnable` in the constructor.
- To start the thread use the method `start()`; it calls the `run()` method in the runnable.

Example

See the ManyThreads program to illustrate the constructs.

Another example

The prime calculator

We input a number N and get the prime number of order N. We use an extra button to test whether the UI is reactive even when calculating large prime numbers.

Calculating in the same thread

What we want to do when a number is given

```
int nr = Integer.parseInt(edittext.getText().toString());

long prime = primeNr( nr );

showtext.setText(""+ prime);
```

Place this code in the `OnKeyListener` for the `EditText`.

For large values it will make the UI unusable: the calculation takes a long time and the main thread cannot take care of other events.

Starting another thread to calculate

What we could do when a number is given

```
new Thread(new Runnable() {
  public void run() {
   int nr = Integer.parseInt(edittext.getText().toString());
   final long prime = primeNr( nr );
   showtext.setText(""+prime);
  }
}).start();
```

Place this code in the `OnKeyListener` for the `EditText`.

The main UI thread and this new worker thread take turns to execute.

Starting another thread to calculate

But it does not work!

We are not allowed to update the UI from other threads!

What we could do when a number is given

```
new Thread(new Runnable() {
  public void run() {
   int nr = Integer.parseInt(edittext.getText().toString());
   final long prime = primeNr( nr );
   showtext.setText(""+prime);
  }
}).start();
```

Posting to the UI thread

Posting *runnables*

Views have a couple of methods that allow you to ask the UI thread to run some code:

```
public boolean post (Runnable action)
public boolean postDelayed (Runnable action,
                            long delayMillis)
```

The prime calculator again

What we do when a number is given

```
new Thread(new Runnable() {
  public void run() {
   int nr = Integer.parseInt(edittext.getText().toString());
   final long prime = primeNr( nr );
   showtext.post(new Runnable() {
       public void run() {
          showtext.setText(""+ prime);
       }
   });
  }
}).start();
```

Services

Applications might need to do work even when the user is not interacting with the app.

Services: to be created and started from other components by passing Intents.
Run in the background and do not provide a UI.
May generate Notifications to start an Activity (with a UI)

Services or worker threads?

If you need to perform work outside your main thread, but only while the user is interacting with your application, then you should probably instead create a new thread

The echo client app

The launcher Activity

1. A button to start a Service to handle a TCP connection.
2. The Activity finishes directly after calling the Service.
3. An Intent is passed to the method that starts the service.
4. All this is done in the listener for the button.

The Service

1. Runs in the main thread.
2. We have to define the methods that are used by the system:
   - `void onCreate()`
   - `int onStartCommand(Intent i, int flags, int id)`
   - `void onDestroy()`
3. Must be terminated explicitely
   - `stopSelf`
   - or `stopService(Intent i)`

Services and threads

Activities and Services of an app run in the same main thread. If we want to do things in other threads we have to do it explicitly.

We would also like the worker thread of a Service to be very much like the main thread: doing nothing but waiting for messages to work on.

Loopers, Handlers, HandlerThreads

1. Every thread in android can be associated with a Looper (listens to messages)
2. We can associate Handlers to Loopers: they can receive messages that are put in a queue and dealt with in order.
3. HandlerThreads are already associated to a looper.

A Service with a ServiceHandler

The following is the program structure we suggest for a Service that can be asked to do several things.

Define a ServiceHandler inside your Service class

```
private final class ServiceHandler extends Handler{
  public ServiceHandler(Looper looper){
    super(looper);
  }
  // override handleMessage:
  public void handleMessage(Message msg){
    // Normally we would do some work here!
    // switch on  msg.what (integer)
    // to distinguish between different things to do!
  }
}
```

A Service with a ServiceHandler (ctd.)

onCreate starts a HandlerThread and associates a ServiceHandler to its Looper

```
public class TheService extends Service{
  private Looper mServiceLooper;
  private ServiceHandler mServiceHandler;

  public void onCreate() {
    HandlerThread thread =
      new HandlerThread("TheServiceWorkerThread",
                        Process.THREAD_PRIORITY_BACKGROUND);
    thread.start();
    mServiceLooper = thread.getLooper();
    mServiceHandler = new ServiceHandler(mServiceLooper);
}
```

A Service with a ServiceHandler (ctd.)

onStartCommand just sends messages to the ServiceHandler

```
public class TheService extends Service{
  public int onStartCommand(Intent intent,
                            int flags,
                            int startId) {
    Message msg = mServiceHandler.obtainMessage();
    msg.what = intent.getExtras().getString("WhatToDo"));
    mServiceHandler.sendMessage(msg);
    return START_STICKY;
  }
}
```

How does a Service start an Activity?

Services that have done what was required of them and want the app to start an Activity to interact with the user should not start the Activity themselves! (the user might be using some other app!).

Instead they should produce a Notification that the user can select in order to start an Activity.

An application with two Activities and a Service

Check the code we distribute with this lecture!