
# Android Application Components: Lifecycle and State

Mobile and Ubiquitous Computing

MEIC/MERC 2015/16

Nuno Santos

# 1. APPLICATION COMPONENTS

# Android Components

- Apps are built out of four types of components:
  - Activity, Service, Broadcast Receiver, and Content Provider


- Components communicate through Intents

# Activity

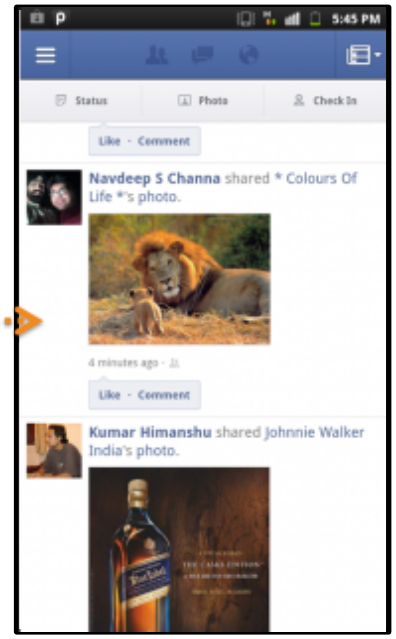
- A typical Android app consists of one or more activities
- Launching the app results in executing one predefined activity (called main activity)
- An activity shows a single visual user interface (GUI)
- An activity may transfer control and data to another activity through messages called intents
- Control and data transfer may occur between activities of different apps

# Activity Example

- Facebook App:
  - Activity A: allows user to login on Facebook
  - Activity B: displays user's Facebook wall

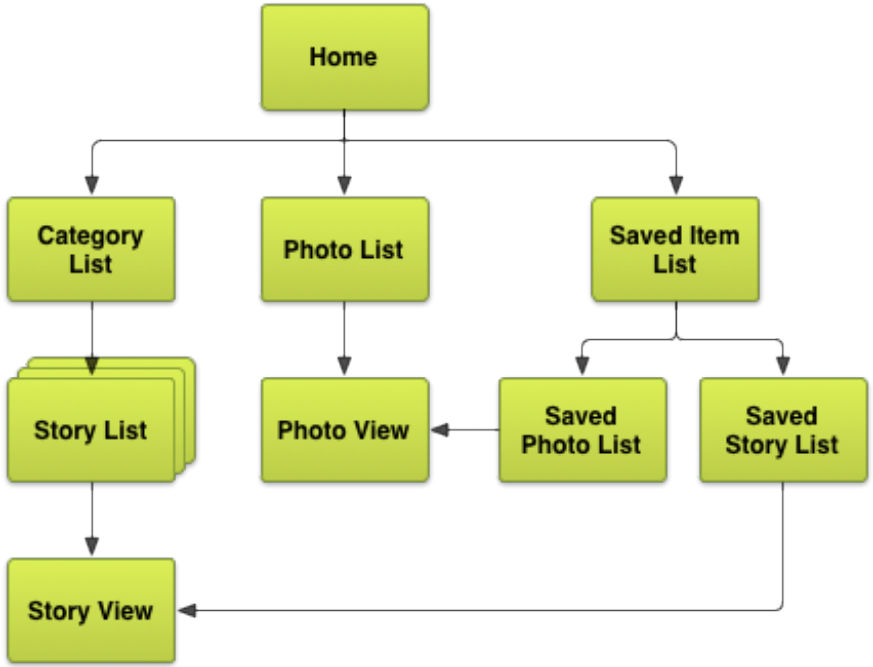
**Activity A**


**Activity B**



# Activity Wireframe

- Screen map of an app with multiple activities
  - Useful to draw before starting to implement the app!



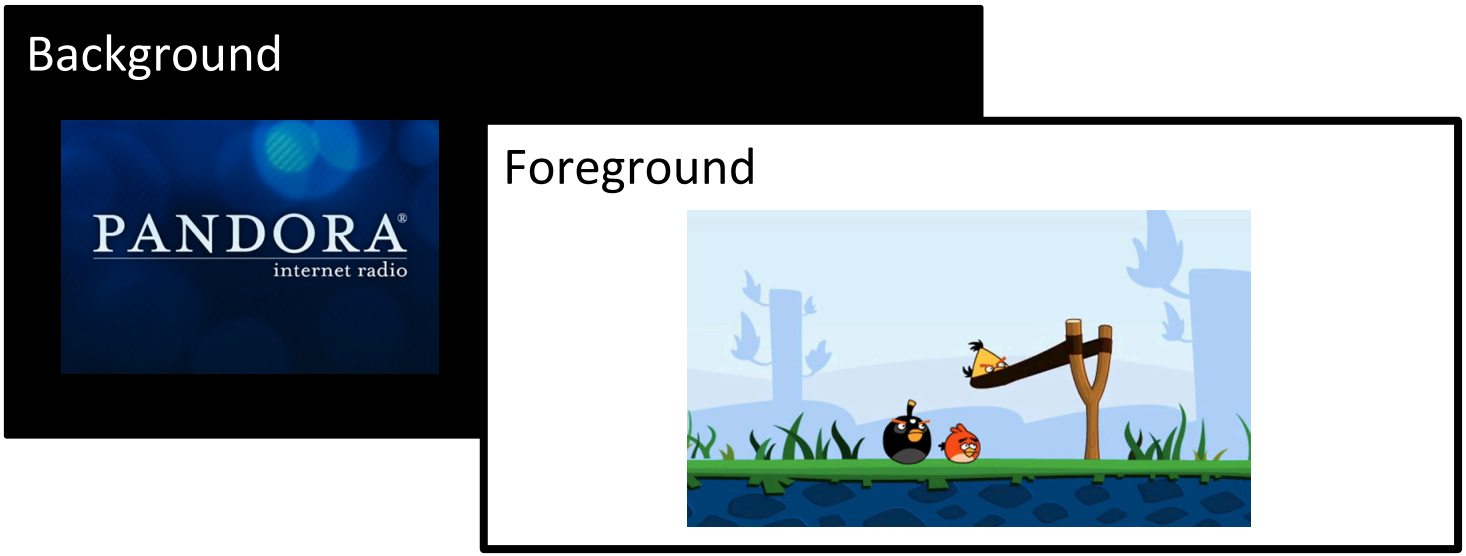

Taken from: http://developer.android.com/training/design-navigation/wireframing.html

# Service

- Services are a special type of activity without visual user interface
- Services run in background (usually for indefinite period of time)
- Applications start their own services or connect to services already active

# Service Example

- Background music
  - A music service (Pandora Radio) runs in background
  - Music heard while other GUIs shown on screen



# Broadcast Receiver

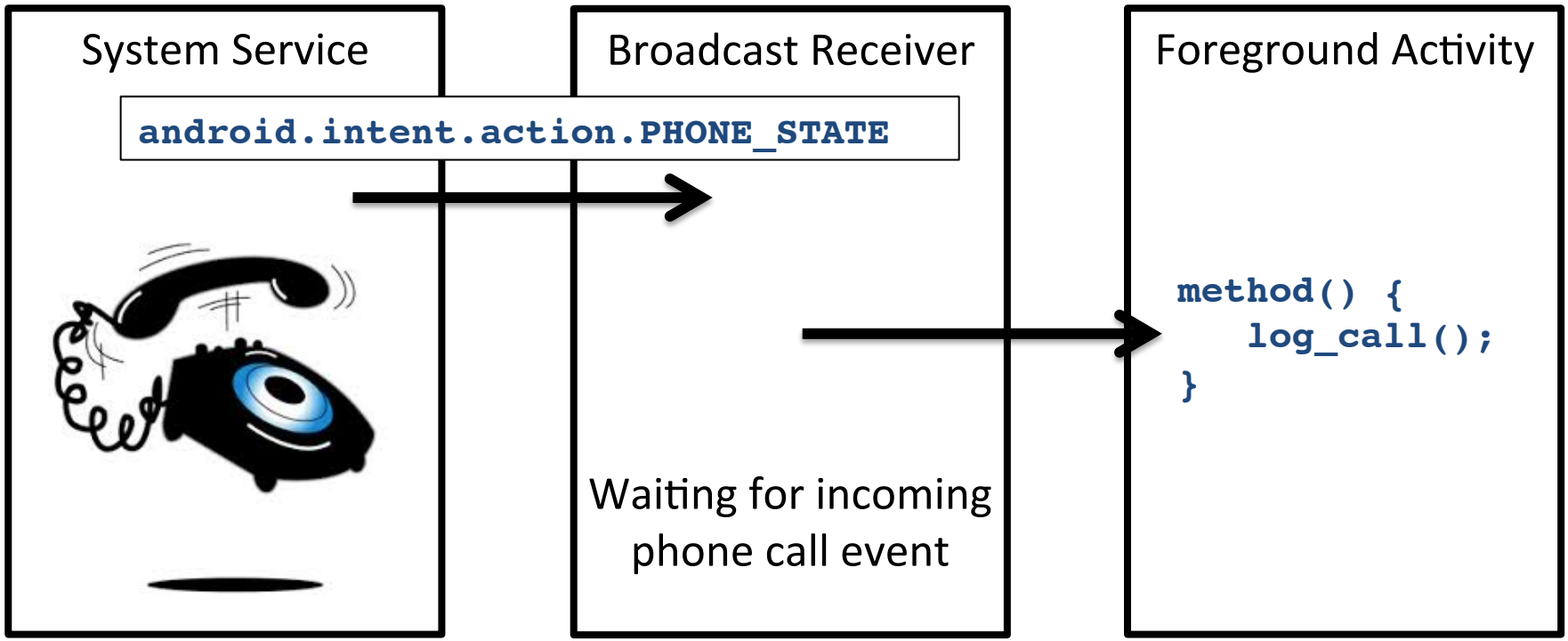
- Is a dedicated listener that waits for system-wide or locally transmitted messages
- Does not display a user interface
- Registered with the system by means of a filter acting as a key
- Activated when the broadcasted message matches the key
- Could respond, e.g., by executing specific activity, or use the notification mechanism to request the user's attention

# Broadcast Event List

- Full list available in sdk/platforms/android-17/data/broadcast_actions
  - `android.app.action.ACTION_PASSWORD_CHANGED`
  - `android.app.action.ACTION_PASSWORD_EXPIRING`
  - `android.app.action.ACTION_PASSWORD_FAILED`
  - `android.app.action.ACTION_PASSWORD_SUCCEEDED`
  - `android.app.action.DEVICE_ADMIN_DISABLED`
  - `android.app.action.DEVICE_ADMIN_DISABLE_REQUESTED`
  - `android.app.action.DEVICE_ADMIN_ENABLED`
  - `android.bluetooth.a2dp.profile.action.CONNECTION_STATE_CHANGED`
  - `android.bluetooth.a2dp.profile.action.PLAYING_STATE_CHANGED`
  - `android.bluetooth.adapter.action.CONNECTION_STATE_CHANGED`
  - `android.bluetooth.adapter.action.DISCOVERY_FINISHED`
  - `android.bluetooth.adapter.action.DISCOVERY_STARTED`
  - `android.bluetooth.adapter.action.LOCAL_NAME_CHANGED`
  - `android.bluetooth.adapter.action.SCAN_MODE_CHANGED`
  - `android.bluetooth.adapter.action.STATE_CHANGED`
  - …

# Broadcast Receiver Example

- Log incoming phone calls



# Content Provider

- Data-centric service that makes persistent datasets available to any number of applications
- Common global datasets include: contacts, pictures, messages, audio files, emails
- Datasets usually stored in an SQLite database
- Content provider offers a standard set of “database-like” methods to enable other apps to retrieve, delete, update, and insert data items

# Decide If You Need a Content Provider

- You need it if you want to provide one of more of the following features:
  - Offer complex data or files to other apps
  - Allow users to copy complex data from your app into other apps
  - Provide custom search suggestions using the search framework
- You don't need one to use SQLite database if the use is local to your app

# Summary of App Components

Represents single screen
E.g., show email list


Works in background w/o UI
E.g., play bg music

Handles system broadcasts
E.g., take action if battery low

Manage access to data
E.g., manage contact information

# 2. APPLICATION LIFECYCLE

# Application Lifecycle

- Each Android application runs inside its own instance of a Dalvik Virtual Machine (DVM)

- An Android application does not completely control the completion of its lifecycle
  - Hardware resources may become critically low and OS could order early termination of any process

# Component Lifecycle

- Components are different points through which the system can enter an app
- Each component has distinct lifecycle that defines how the component is created and destroyed
- They all follow a master plan that consists of:
  1. Begin: respond to request to instantiate them
  2. End: when instances are destroyed
  3. In between states: that depend on component type

# Activity Lifecycle

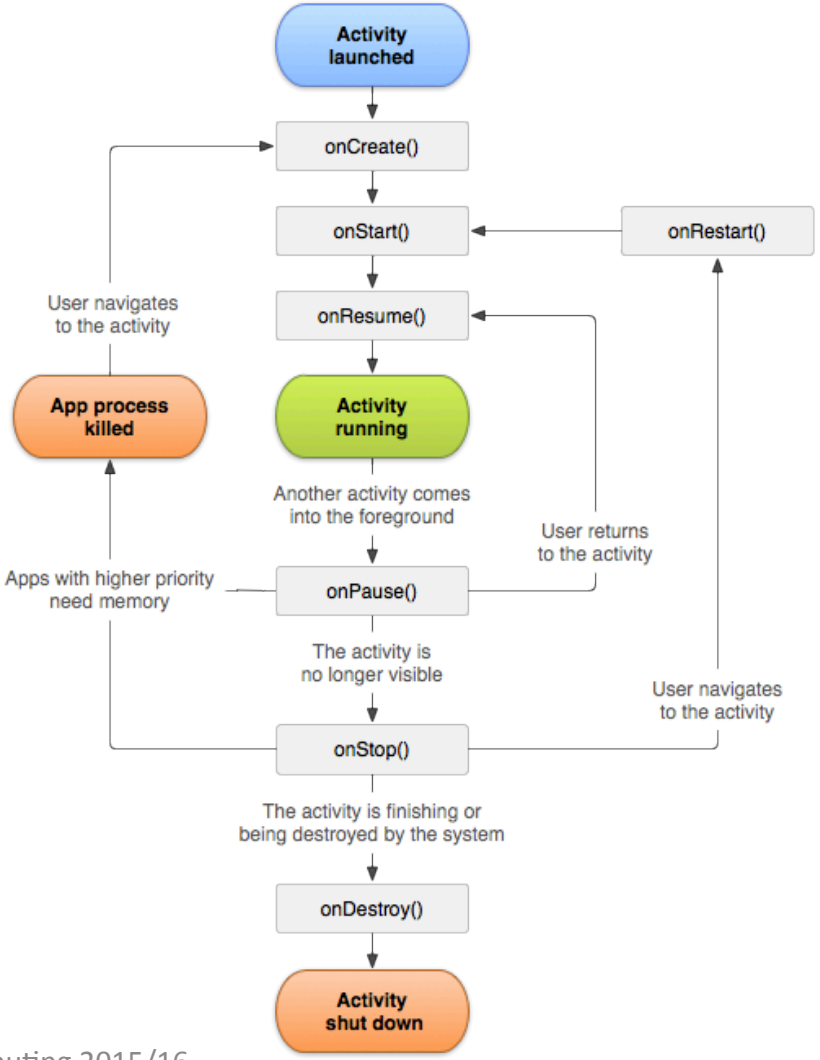
- An activity can exist in essentially three states:

| | |
|---|---|
| **Resumed (aka Running)** | The activity is in the foreground of the screen and has user focus |
| **Paused** | Another activity is in the foreground and has focus, but this is still visible (e.g., transparently); it is alive but can be killed |
| **Stopped** | The activity is completely obscured by another activity (it is now in background); it can be killed if memory is needed |

- If an activity is paused or stopped, the system can drop it from memory
  - When the activity is opened again, it must be created all over

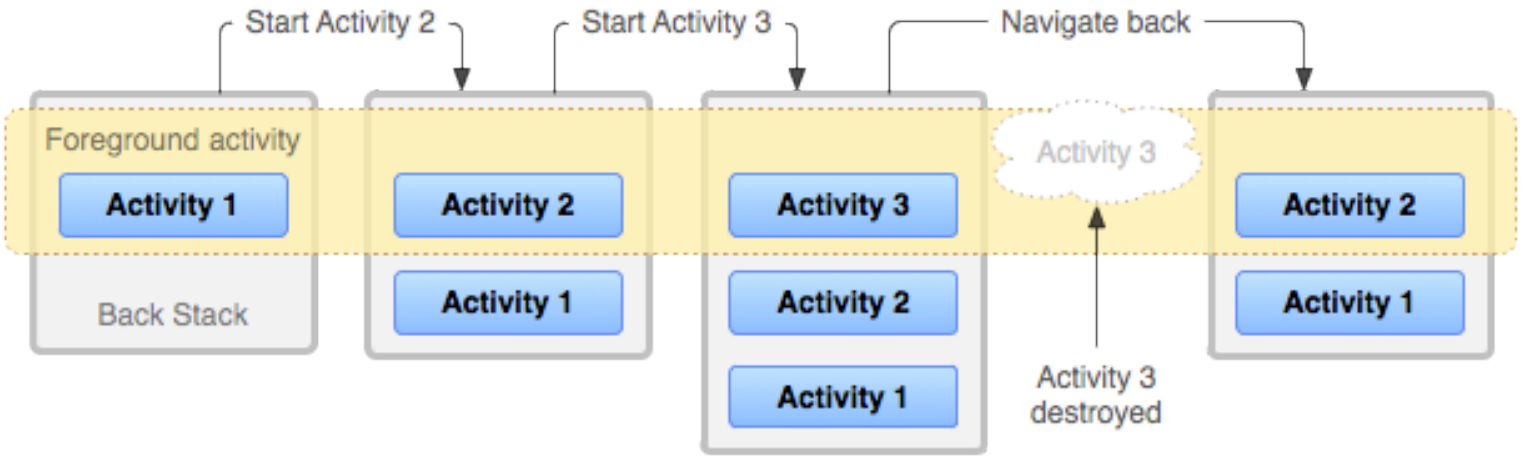
# Activity Lifecycle Events

- When progressing from one state to the other, the OS notifies the app of the changes by issuing calls to the following transition methods:
  - `onCreate`
  - `onStart`
  - `onRestart`
  - `onResume`
  - `onPause`
  - `onStop`
  - `onDestroy`



# Back Stack

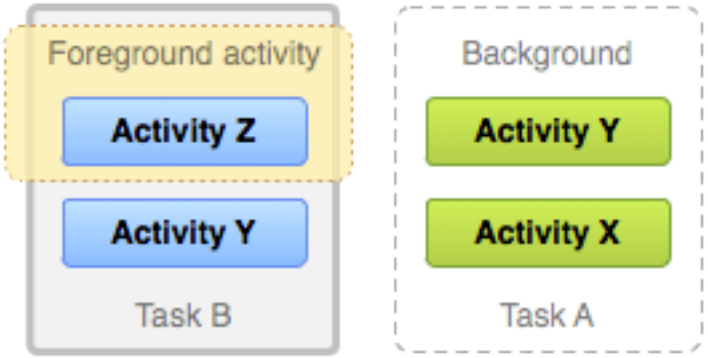
- Activities are scheduled using a stack named back stack




  - When a new activity is started, it is placed on top of the stack and becomes the running activity
  - Previous activity is pushed-down one level in the stack and changes to paused / stopped state
  - If the user presses the *Back Button* or the foreground activity terminates, the next activity is resumed

# Tasks

- The system can manage independent back stacks as tasks
- A task is created first time user touches app icon in Home screen
- A task can be entirely moved to the background
  - E.g., when user presses the Home button



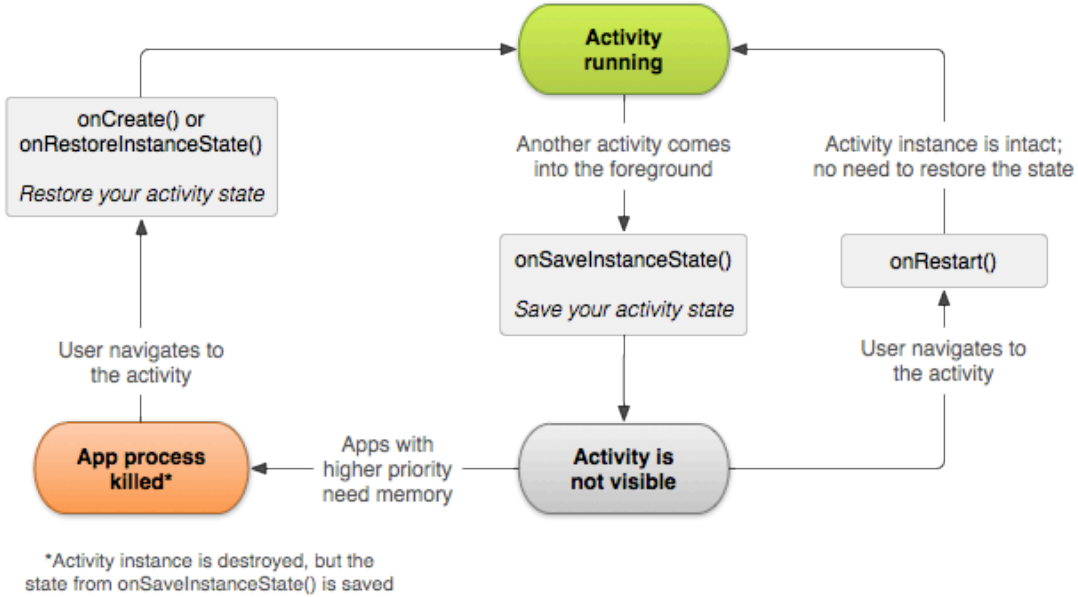

- Activities in a task's back stack could belong to different apps
  - E.g., sub-activity for sending an email

# 3. APPLICATION STATE

# Activity State

# Saving Activity State

- When the system destroys an activity, the system must recreate the Activity object if the user navigates back to it
- To preserve activity state, implement **`onSaveInstanceState`**




http://developer.android.com/training/basics/activity-lifecycle/recreating.html

# Recovering Activity State

- `protected void onCreate(Bundle savedState)`
  - 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
  - It also provides you with a Bundle containing the activity's previously frozen state, if there was one

- `protected void onPause()`
  - Called when the system is about to start resuming a previous activity
  - It is typically used to commit unsaved changes to persistent data, stop animations, release system resources, etc.
  - Implementations of this method must be very quick because the next activity won't be resumed until this method returns

Taken from: http://developer.android.com/reference/android/app/Activity.html

# Cross-Component Communication

# Cross-Component Communication: Intents

- Intent is a messaging object to request an action from another app component

- Fundamental use-cases:

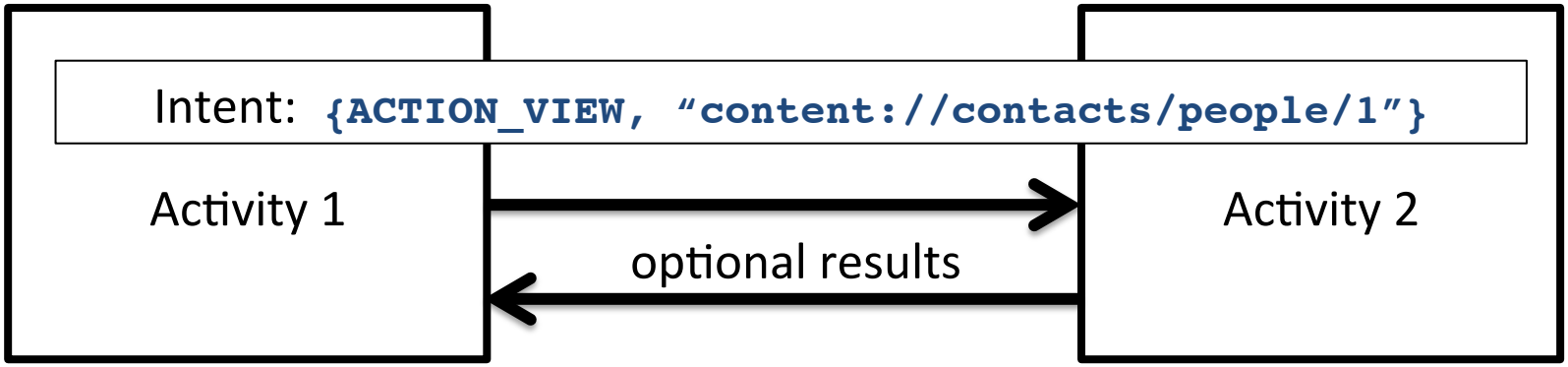
| | |
|---|---|
| Start an activity | `startActivity(intent)` |
| Start a service | `startService(intent)` |
| Deliver a broadcast | `sendBroadcast(intent)` |

Taken from: https://developer.android.com/guide/components/intents-filters.html

# Main Arguments of Intents

1. **Action**: The built-in action to be performed, such as `ACTION_VIEW`, or user-created-action

2. **Data**: The primary data to operate on, such as a phone number to be called (expressed as a URI)

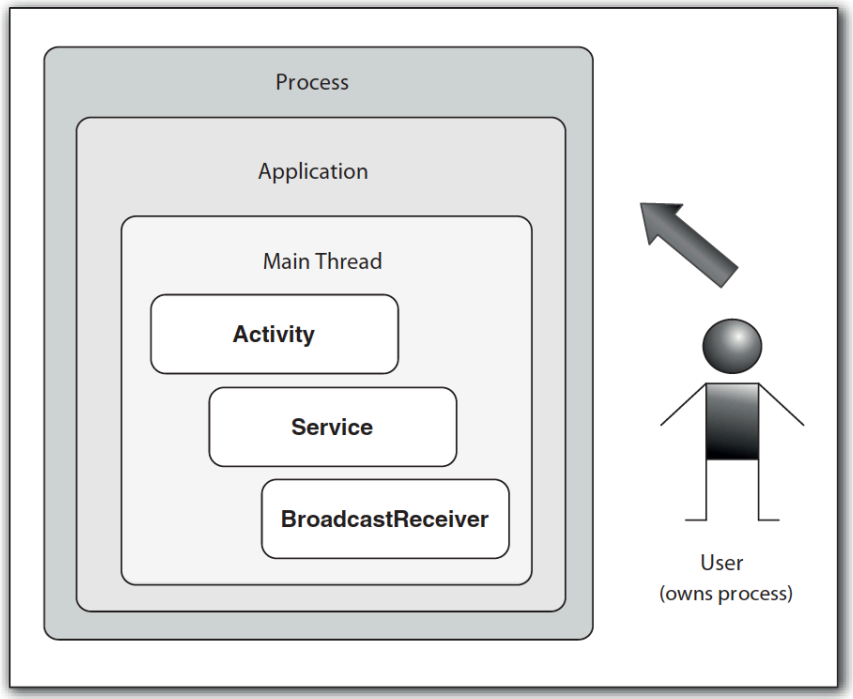



Taken from: https://developer.android.com/reference/android/content/Intent.html

# Application Context

# Application Global State

- Every Android application is hosted by an Application object
  - We use the Application object to share global state
  - Each app runs in its own process with its own ID and main thread



# Using the Application Context

- Either use the default application context or create your own:
  - Create a class for holding the app shared state; this class must extend from class `android.app.Application`

```
package pt.ulisboa.tecnico.cmov.globalvariable;

import android.app.Application;

public class GlobalClass extends Application {

    private String name;

    public String getName() {

        return name;
    }

    public void setName(String aName) {

       name = aName;

    }
}
```

# Using the Application Context

- Declare the new context class in the manifest file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package=”pt.ulisboa.tecnico.cmov.globalvariable"
    ... >

    <uses-sdk
        ... />

    <application
        android:name="pt.ulisboa.tecnico.cmov.globalvariable.GlobalClass"
        ... >
        <activity
            ... >
            ...

        </activity>

        ...
    </application>

</manifest>
```

# Using the Application Context

- Access the application context object from any component of the application
  - To write:

```
// Obtain reference to application context
GlobalClass globalVariable = (GlobalClass) getApplicationContext();

// Set name in global/application context
globalVariable.setName("Android Example context variable");
```

  - To read:

```
// Obtain reference to application context
GlobalClass globalVariable = (GlobalClass) getApplicationContext();

// Get name from global/application context
String name  = globalVariable.getName();
```

# Application Context Lifecycle Methods

- onCreate: called when the Application is started
- onLowMemory: called when the system requests that apps try to clean up what they can
- onTerminate: *sometimes* called when the Application is stopped
- onConfigurationChanged: called when the device Configuration changes while the app is running

# Useful Pointers

- Service lifecycle
  - http://developer.android.com/guide/components/services.html
- Broadcast receiver lifecycle
  - http://developer.android.com/reference/android/content/BroadcastReceiver.html
  - http://www.grokkingandroid.com/android-tutorial-broadcastreceiver/
- Content provider lifecycle
  - http://developer.android.com/guide/topics/providers/content-providers.html