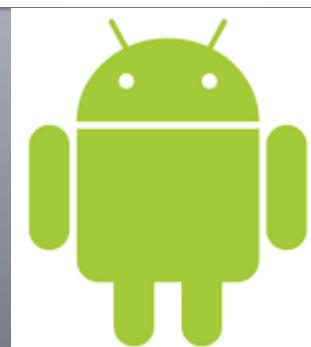


# Développement Applications Mobiles I

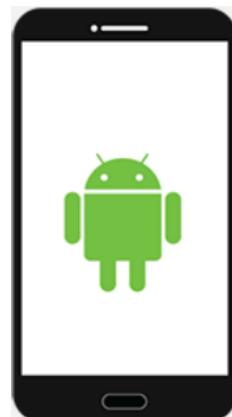
Filière: D-LSI ADBD & MM

Enseigné par : **Rim Walha**



# Objectifs du cours

- Acquérir les concepts de base d'une application mobile
- Se familiariser avec l'environnement Android Studio
- Gérer les différents éléments d'une application Android
- Développer des applications Android



# Pré-requis

- Programmation orientée objet
- Connaissance du langage **JAVA**
- Un smartphone Android **n'est pas nécessaire pour ce cours.**
- Google fournit un **émulateur Android** qui vous permettra de faire tourner sur votre PC un téléphone ou une tablette **virtuelle**.

# Contenu du cours

- **Chapitre I:** Introduction au développement des applications mobiles
- **Chapitre II:** La plateforme Android: Vue d'ensemble
- **Chapitre III:** Développer une Application Android
- **Chapitre IV:** Ressources et Activités d'une Application Android
- **Chapitre V:** Les intentions

# **Chapitre I**

**Introduction au développement  
des applications mobiles**

# Plan du chapitre I

- Contexte
- Technologies mobiles actuelles
- Application mobile
- Plateformes mobiles
- Contraintes de développement des applications mobiles

# Contexte

## Smartphone



- “A smartphone is a mobile phone with **more advanced computing capability and connectivity** than a feature phone.”  
[Wikipedia]
  - “A feature phone is a mobile phone which is not considered to be a smartphone due to the lack of several features.”
- Un smartphone : **téléphone mobile intelligent**

# Technologies mobiles actuelles

## Trafic Web par appareil :

JAN  
2020

### SHARE OF WEB TRAFFIC

EACH DEVICE'S SHARE OF TOTAL WEB PAGES SERVED TO INTERNET USERS

Le support mobile est devenu de plus en plus intéressant pour se connecter à internet

MOBILE  
PHONES

LAPTOPS &  
DESKTOPS

TABLET  
COMPUTERS

OTHER  
DEVICES



**53.3%**

DEC 2019 vs. DEC 2018:

**+8.6%**



**44.0%**

DEC 2019 vs. DEC 2018:

**-6.8%**



**2.7%**

DEC 2019 vs. DEC 2018:

**-27%**



**0.07%**

DEC 2019 vs. DEC 2018:

**-30%**

# Application mobile

## Présentation

- C'est un **logiciel** développé pour être exécutable sur un **dispositif mobile** (smartphone, tablette,...)
- **Fonctionnalités des applications mobiles:**
  - Réserver des services,
  - Acheter des produits,
  - Communiquer,
  - Consulter des comptes bancaires,
  - Localiser des endroits, ....



# Application mobile

## Modèles de développement

### Application web

- C'est un site web fonctionnant dans un navigateur web et conçue **spécifiquement pour l'usage à partir d'un terminal mobile.**

### Application native

- C'est une application mobile développée **spécifiquement pour un système d'exploitation** utilisé par les terminaux mobiles.

### Application hybride (ou multiplateforme)

- C'est une **encapsulation** d'une application web dans une application native qui utilise **des vues web** (composants qui affiche les pages web sans barre d'adresse, ni contrôles de navigation contrairement à un navigateur web).

# Plateformes mobiles

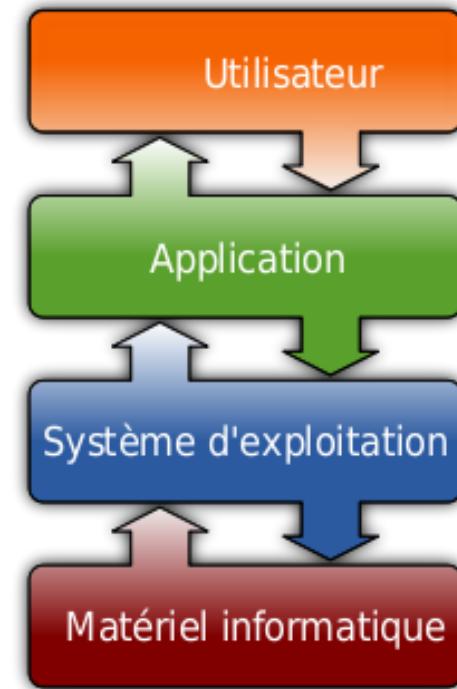
## Système d'Exploitation

### Système d'Exploitation (SE)

- Système d'Exploitation (Operating System): C'est un **ensemble de programmes** qui permettent de faire l'interface entre le matériel informatique et les applications développées.

### Système d'Exploitation mobile

- C'est un système d'exploitation conçu pour fonctionner sur un appareil mobile.



# Plateformes mobiles

## Systèmes d'Exploitation mobiles

### SE mobiles:

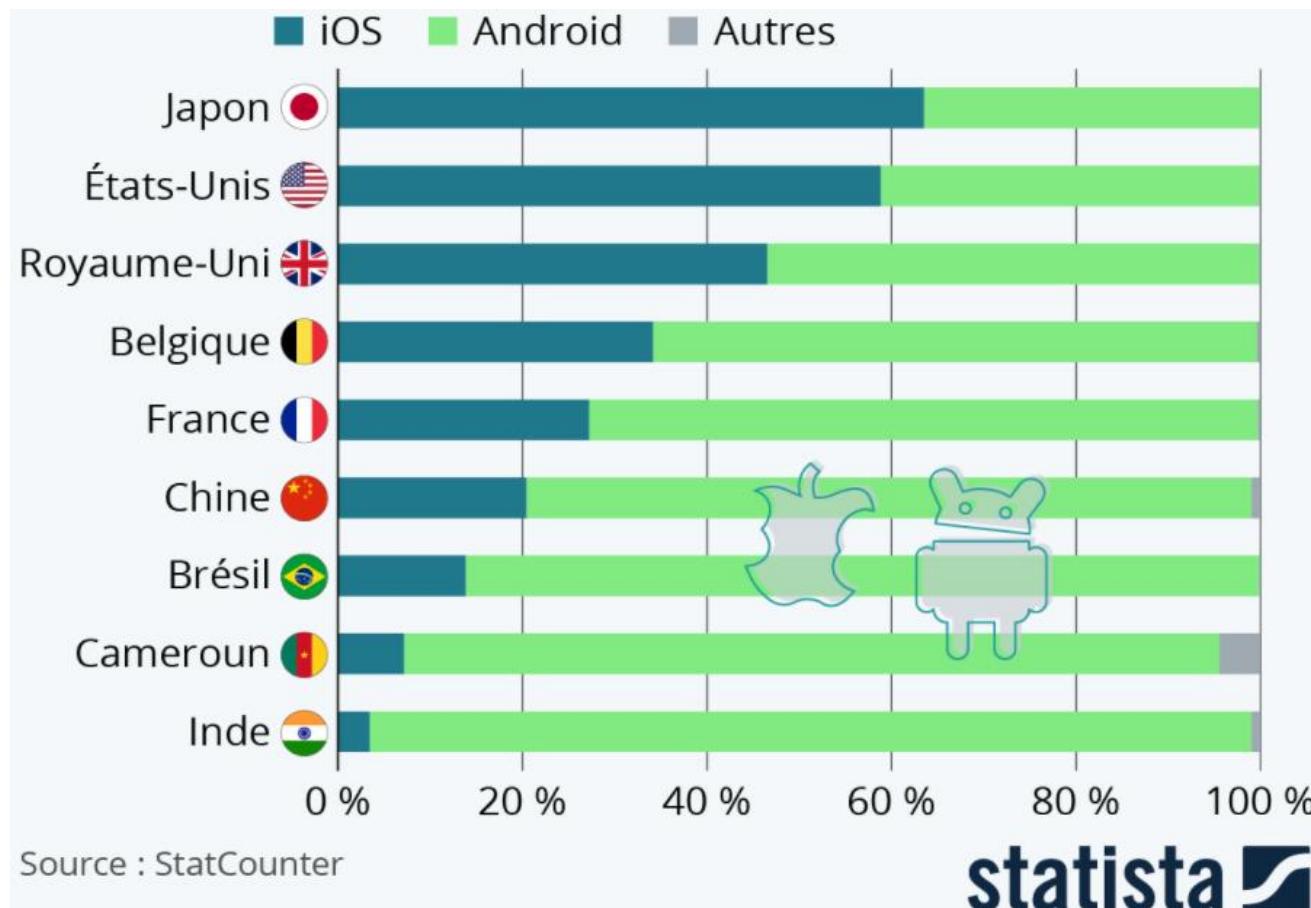
- **Android** de Google
- **iOS** de Apple
- **BlackBerry OS** de Research In Motion
- **Windows Phone** de Microsoft
- **Bada** de Samsung (*même si Samsung utilise aussi Android*)
- **webOS** de HP
- **Symbian OS** de Nokia
- ...



# Plateformes mobiles

## Android et ses concurrents

Parts de marché des **SE mobiles** dans une sélection de pays  
Juillet 2020 selon Statista 2020



# Plateformes mobiles

## Android et ses concurrents

JAN  
2020

### SHARE OF WEB TRAFFIC BY MOBILE OS

SHARE OF WEB PAGE REQUESTS ORIGINATING FROM MOBILE HANDSETS RUNNING DIFFERENT MOBILE OPERATING SYSTEMS



TUNISIA

SHARE OF WEB TRAFFIC  
ORIGINATING FROM  
ANDROID DEVICES



**93.4%**

DEC 2019 vs. DEC 2018:

**+3.4%**

SHARE OF WEB TRAFFIC  
ORIGINATING FROM  
APPLE IOS DEVICES



**5.7%**

DEC 2019 vs. DEC 2018:

**-20%**

SHARE OF WEB TRAFFIC  
ORIGINATING FROM  
SAMSUNG OS DEVICES\*



**0.1%**

DEC 2019 vs. DEC 2018:

**-42%**

SHARE OF WEB TRAFFIC  
ORIGINATING FROM  
OTHER OS DEVICES



**0.8%**

DEC 2019 vs. DEC 2018:

**-67%**

# Plateformes mobiles

## Android et ses concurrents

Android (Google) vs iOS (Apple)



Source: App Annie, spécialiste de l'analyse de données sur les boutiques d'applications

# Contraintes de développement des applications mobiles

- Gardez à l'esprit que les périphériques mobiles ont souvent :
  - Une puissance processeur plus faible
  - Une RAM limitée
  - Des capacités de stockage permanent limitées
  - De petits écrans avec de faibles résolutions
  - Des taux de transfert plus lents
  - Des connexions réseau moins fiables
  - Des batteries à autonomie limitée

# **Chapitre II**

## **La plateforme Android: Vue d'ensemble**

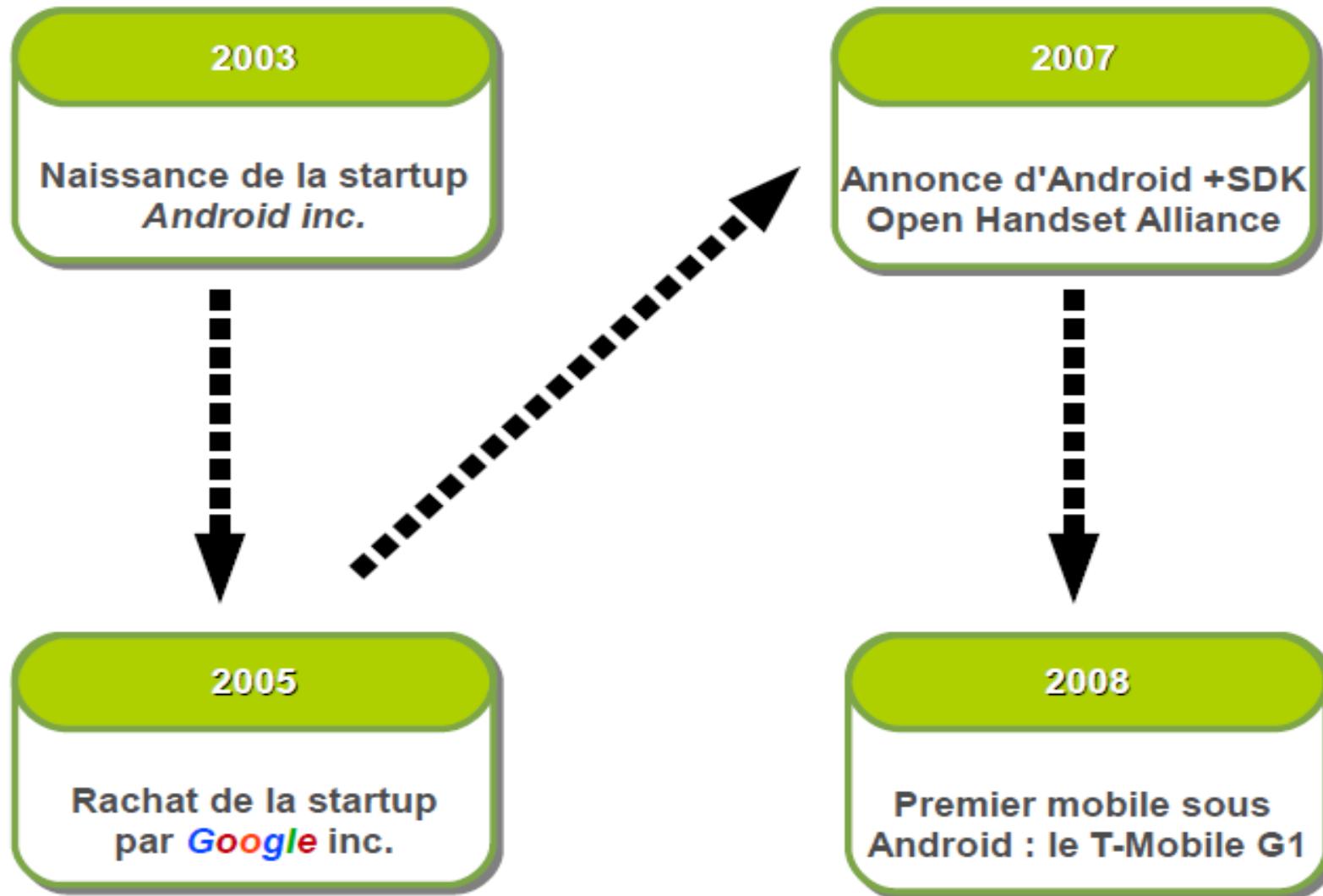
# Plan du chapitre II

- Présentation d'Android
- Android: Bref historique
- Architecture logicielle d'Android
- Points forts d'Android

# Présentation d'Android

- **Système d'Exploitation** orienté **dispositifs mobiles**
- SE **open source** (code disponible)
- SE basé sur **le noyau Linux**
- Environnement de développement **gratuit**
  - Programmation en Java ou en langage C,
  - Kit de développement (SDK Android) disponible au lien  
<http://developer.android.com/sdk/index.html>

# Android: Bref historique



# Android: Bref historique

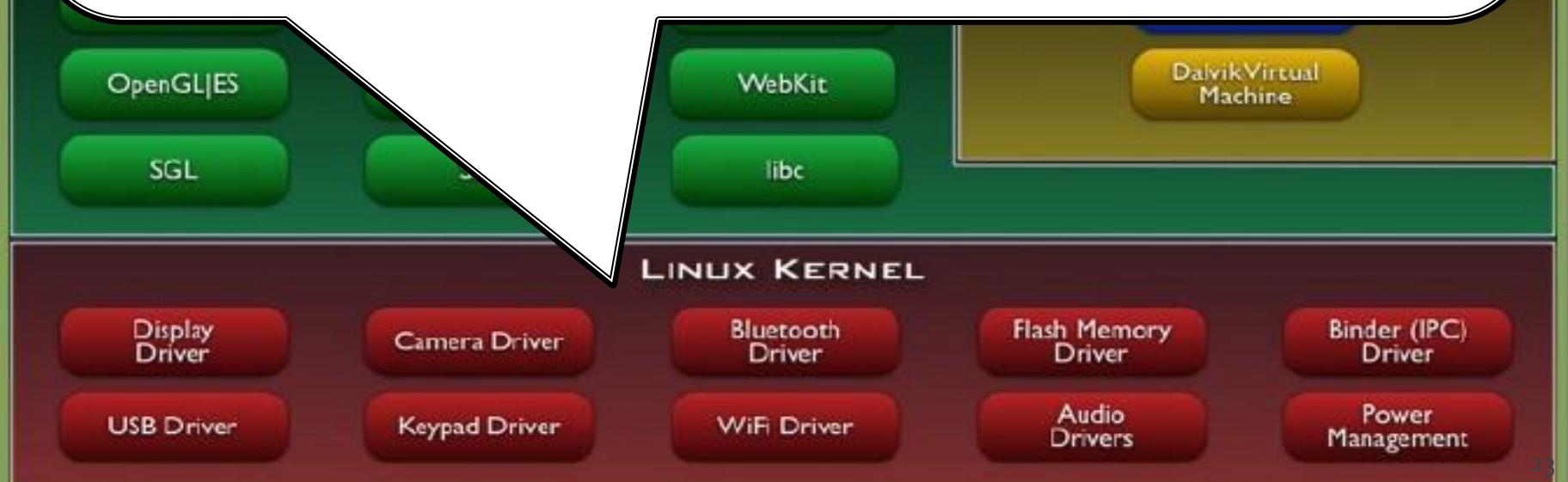


# Architecture logicielle d'Android



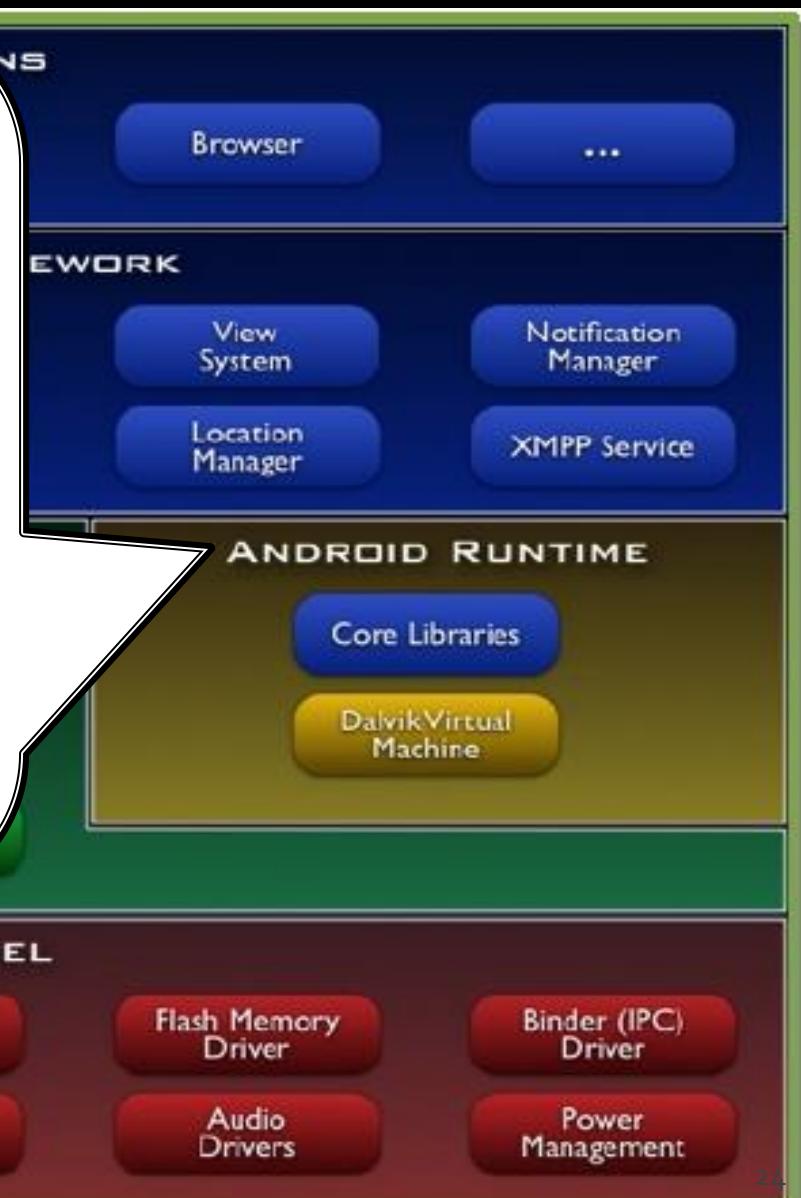
# Architecture logicielle d'Android

- Le noyau est l'élément du SE qui permet de faire **le pont entre le matériel et le logiciel**.
- Offre les **services fondamentaux**: pilotes du hardware, gestion de la mémoire et des processus, sécurité, réseau, ...
- La version du noyau utilisée (**Linux 2.6**) est conçue spécialement pour **l'environnement mobile**, avec une gestion particulière de la batterie et de la mémoire.



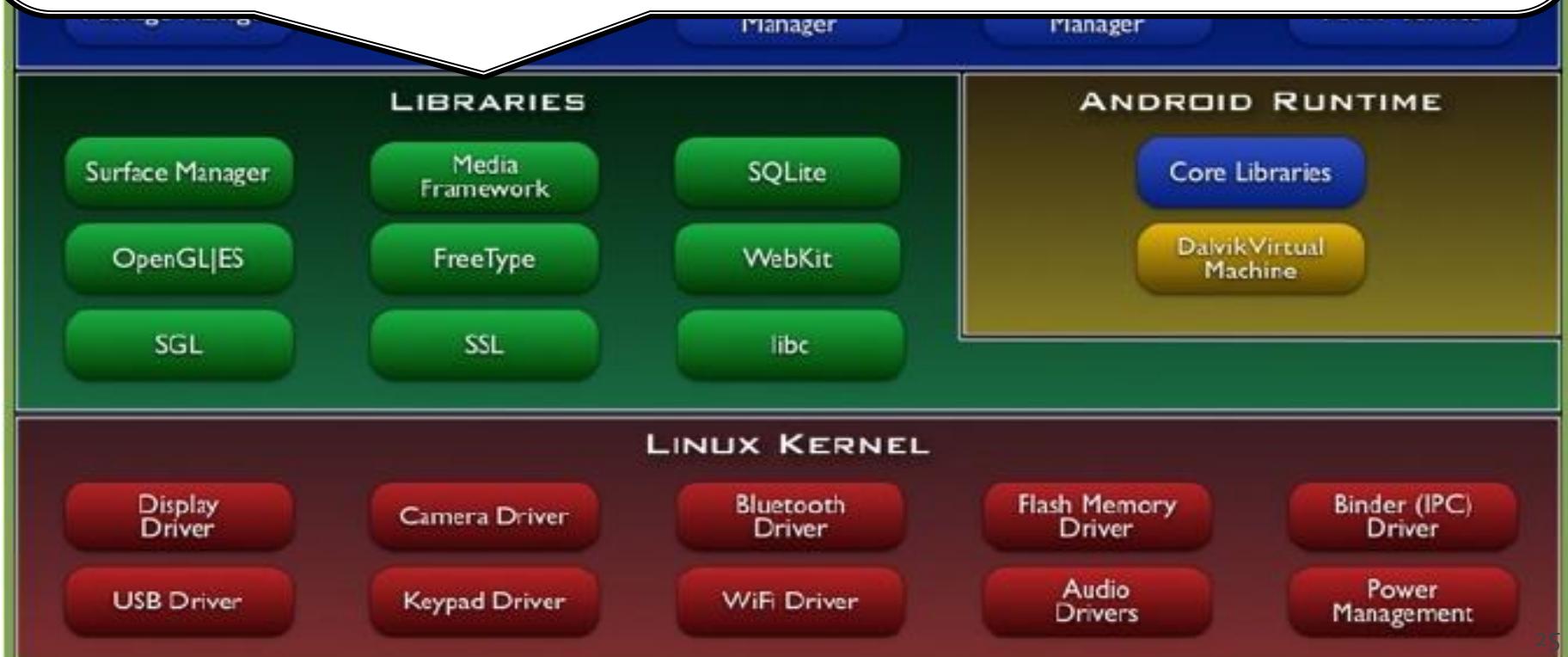
# Architecture logicielle d'Android

- C'est le **moteur d'exécution d'Android** contenant **des librairies de base** et la **machine virtuelle « Dalvik »** permettant l'exécution des applications.
- « Dalvik »: une **machine virtuelle Java adaptée** pour mieux gérer les ressources du système mobile (allocation réduite de la mémoire lors de l'exécution d'une application, usage réduit de la batterie qu'une machine virtuelle Java).



# Architecture logicielle d'Android

- Ces librairies, écrites en C/C++, fournissent les **fonctionnalités de bas niveau**:
  - ✓ gestion de l'affichage via **Surface Manager**,
  - ✓ gestion des BD via **SQLite**,
  - ✓ lecture audio/vidéo via **Media Framework**,
  - ✓ affichage 2D et 3D via **SGL et OpenGL**,
  - ✓ navigation sur internet via **SSL et Webkit**, ...



# Architecture logicielle d'Android

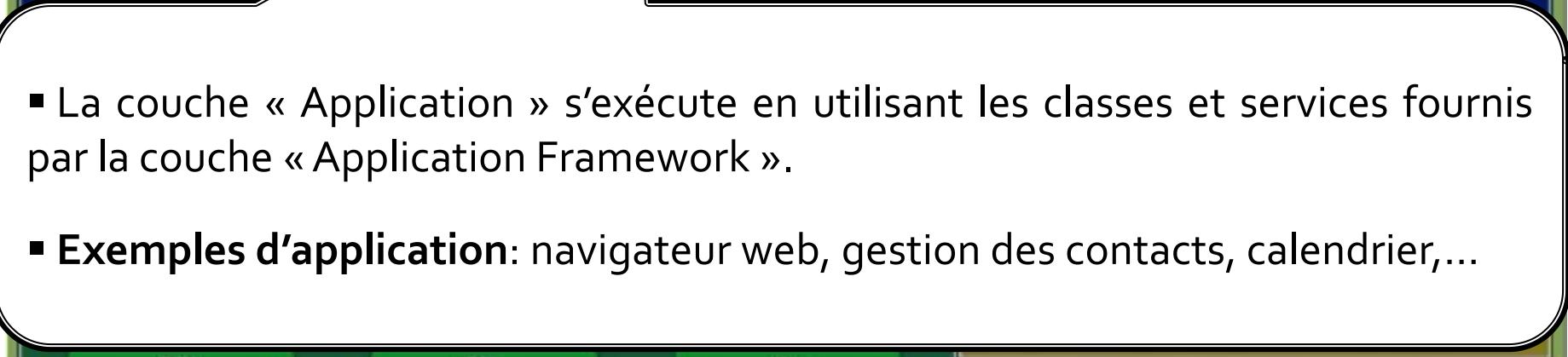


- Fournit aux développeurs des **interfaces de programmation applicative**, désignées **API** (Application Programming Interface), permettant la création d'applications Android riches.
- **Exemples de services offerts par ces API:**
  - gestion du cycle de vie, des interfaces utilisateur et des ressources média d'une application,
  - gestion des données entre applications,
  - accès aux interfaces matérielles (téléphonique, GPS, bluetooth, Wifi, ...).

# Architecture logicielle d'Android



- La couche « Application » s'exécute en utilisant les classes et services fournis par la couche « Application Framework ».
- **Exemples d'application:** navigateur web, gestion des contacts, calendrier,...



# Points forts d'Android

- **Android est open-source et gratuit**
  - Noyau Linux
  - Code source disponible qu'on peut le consulter, le télécharger, l'adapter, ... (contrairement à Windows ou Mac OS)
  - Associé à un large ensemble de bibliothèques open-source
- **Développement est accessible**
  - SDK complet fourni → facilité de développement
  - Plusieurs APIs sont fournies en vue d'accélérer le développement.
- **Système est évolutif**
  - Comme c'est un système ouvert, il est donc facilement portable d'un appareil à un autre : sur nos smartphones, nos tablettes, mais aussi sur d'autres appareils électroniques du quotidien et même nos véhicules.

# **Chapitre III**

## **Développer une Application Android**

# Plan du chapitre III

- Eléments d'une application Android
- Environnements de développement
- Développer avec Android Studio
- Structure d'un projet Android
- Première Application Android
- Compilation et déploiement d'un projet Android
- Exécution de l'application

# Eléments d'une application Android

## Aperçu général

- Une application Android peut être composée des éléments suivants:



- **Activité**

Programme qui gère une **interface graphique**



- **Service**

Programme qui fonctionne **en tâche de fond** sans interface



- **Fournisseur de contenus**

**Partage d'informations** entre applications



- **Ecouteur d'intention diffusées**

Permet à une application de **récupérer des informations générales** (extinction de l'écran, réception d'un SMS, batterie faible, ...)



- **Eléments d'interaction**

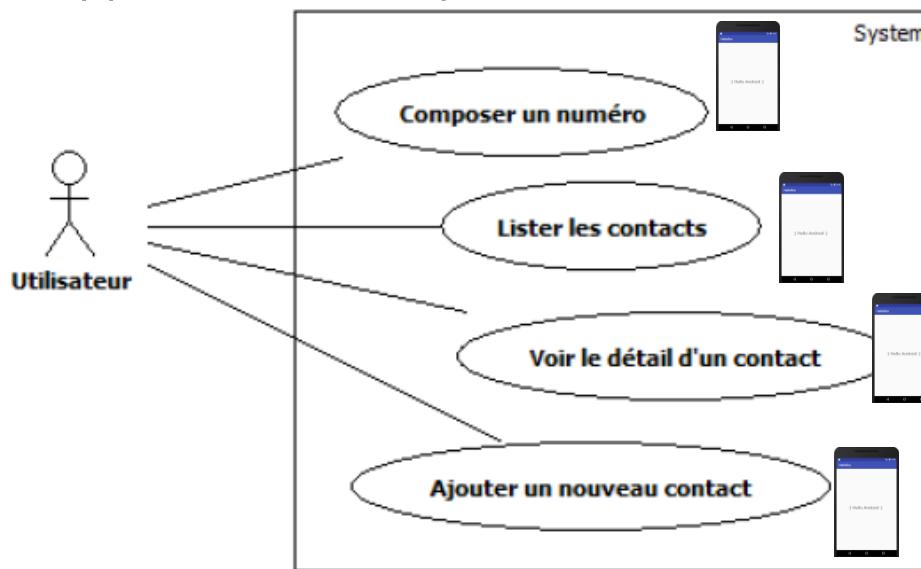
- **Intention** : permet à une application de **chercher un savoir-faire**

- **Filtre d'intentions** : permet à une application d'**indiquer ce qu'elle sait faire** et donc de **choisir la meilleure application** pour assurer un savoir-faire

# Eléments d'une application Android

## Activité

- Activité ≈ interface graphique pour l'utilisateur
- Incarne souvent un cas d'utilisation (use case UML)
  - Exemple : application de téléphonie



- Une application est formée de  $N$  activités
- Une activité hérite de la classe `android.app.Activity`

# Eléments d'une application Android

## Service

- Service ≈ activité sans interface graphique pour l'utilisateur.
- Logiciel autonome prévu pour durer (**activité tâche de fond** ).
- Exemples :
  - Service permettant d'écouter la musique
  - Service pour rechercher des données sur le réseau
- Un service hérite de la classe `android.app.Service`

# Eléments d'une application Android

## Fournisseur de contenu

- Un fournisseur de contenu (**content provider**) permet le **partage des données** au sein ou entre applications.
- Exemples :
  - Accès aux contacts stockés dans le téléphone
  - Accès à l'agenda
  - Accès aux photos
- Ce composant propose un contrôle sur la façon dont on accèdera aux informations stockées sur le terminal.
- Il expose les données via une **URI** (schema://host:port/path) dont le schéma dédié est 'content'
  - **content://sms/inbox/25**
  - **content://com.whatsapp.provider.media/item/12345**
- Un fournisseur de contenu hérite de la classe **android.content.ContentProvider**

# Eléments d'une application Android

## Ecouteur d'intentions diffusées

- Un écouteur d'intention diffusées (**BroadcastReceiver**) est un composant à **l'écoute d'informations** qui lui sont destinées.
- Un tel récepteur indique le type d'informations qui l'intéressent et pour lesquelles il se mettra en écoute.
- Ce composant permet à une application de **récupérer des informations** générales tels que:
  - réception d'un SMS, appel téléphonique entrant, batterie faible, réception de la liste des réseaux Wi-Fi connecté, ...
- Un écouteur d'intention ne nécessite pas une interface graphique
- Un écouteur d'intention diffusées hérite de la classe **android.content.BroadcastReceiver**

# Eléments d'une application Android

## Fichier Manifest

- **Point de départ** de toute application Android
- Chaque projet contient à sa racine un fichier **AndroidManifest.xml**
  - Nomme le paquetage Java de l'application. Ce dernier sert d'identificateur unique de l'application.
  - Permet de déclarer **les composants** de l'application (activités, services, ...) et précise **les relations** entre eux (que fait-on apparaître dans le menu ?, ...)
  - Précise **les permissions** de l'application (droit d'accès à internet, à l'appareil photo, au service de localisation GPS, droit de passer des appels, ...)
  - Déclare le **niveau minimum de compatibilité nécessaire du SDK** pour que l'application fonctionne
  - Déclare les **librairies** utilisées



# Environnements de développement

- Il existe de nombreux IDE pour Android:

- Android Studio
- Eclipse + le plug-in ADT (Android Development Tools)
- Apache Cordova
- Oracle NetBeans
- WinDev Mobile

Les plus utilisées

- Langages de développement:

- Java avec Android SDK (Software Development Kit)
- C++ avec Android NDK (Native Development Kit)

# Développer avec Android Studio

- C'est l'IDE officiel de Google.
- Cet IDE est disponible **gratuitement** sur le lien :  
<https://developer.android.com/studio/>



# Développer avec Android Studio

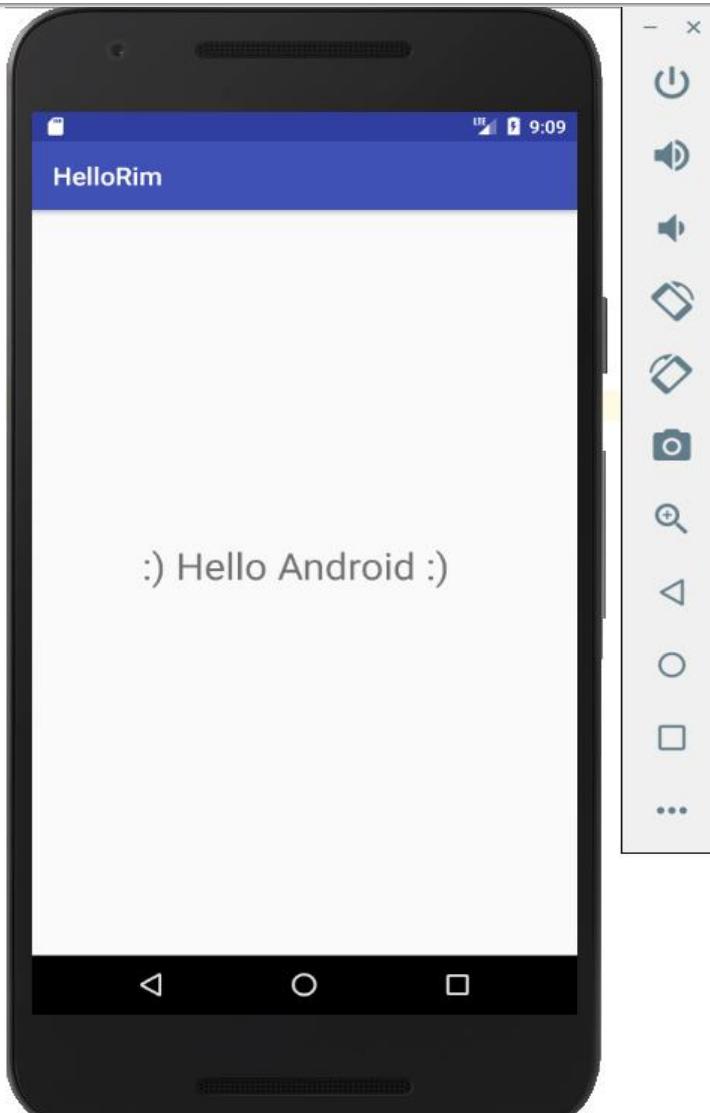
## Aperçu de l'interface

The screenshot displays the Android Studio interface with several key components highlighted:

- Compilation**: Located at the top left of the toolbar.
- Exécution et Debug**: Located at the top right of the toolbar.
- Explorateur**: Project browser on the left.
- Palette des composants graphiques**: A list of graphical components on the left.
- Navigateur des fichiers ouverts**: File browser at the top center.
- Aperçu de l'activité**: Preview of the activity layout in the center.
- Navigation entre vue graphique et xml**: Buttons at the bottom of the preview area.
- Propriétés du composant sélectionné**: Properties panel on the right.
- Liste des composants de l'activité**: Component tree on the right.
- Output**: Logcat window at the bottom.

# Première Application Android

## Aperçu



# Première Application Android

## Code source

Ce fichier contient la description de l'IHM et peut être visualisé en mode **Design**.

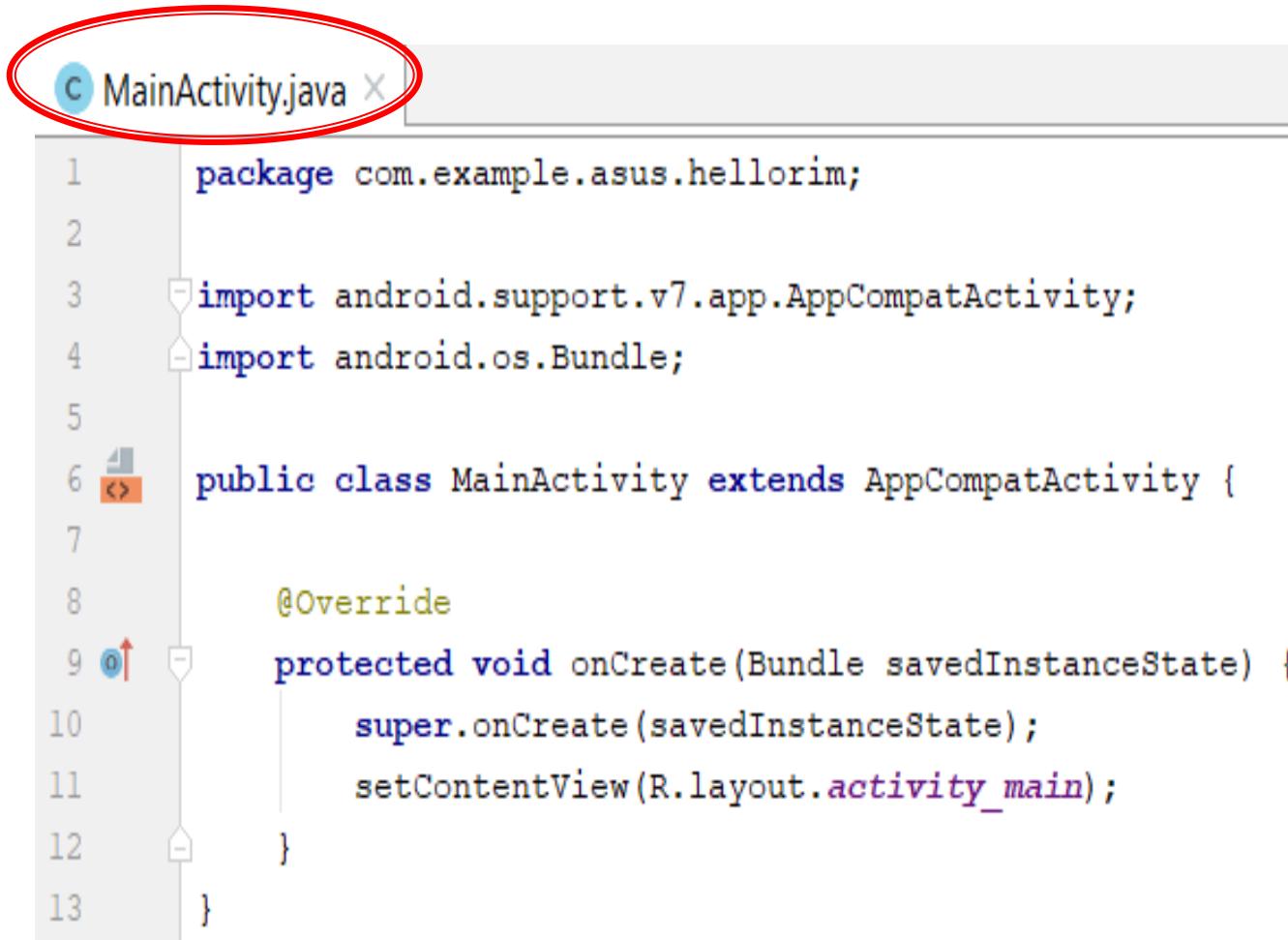
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="245dp"
        android:layout_height="69dp"
        android:text=":) Hello Android :)"
        android:textSize="30sp" />

</android.support.constraint.ConstraintLayout>
```

# Première Application Android

## Code source



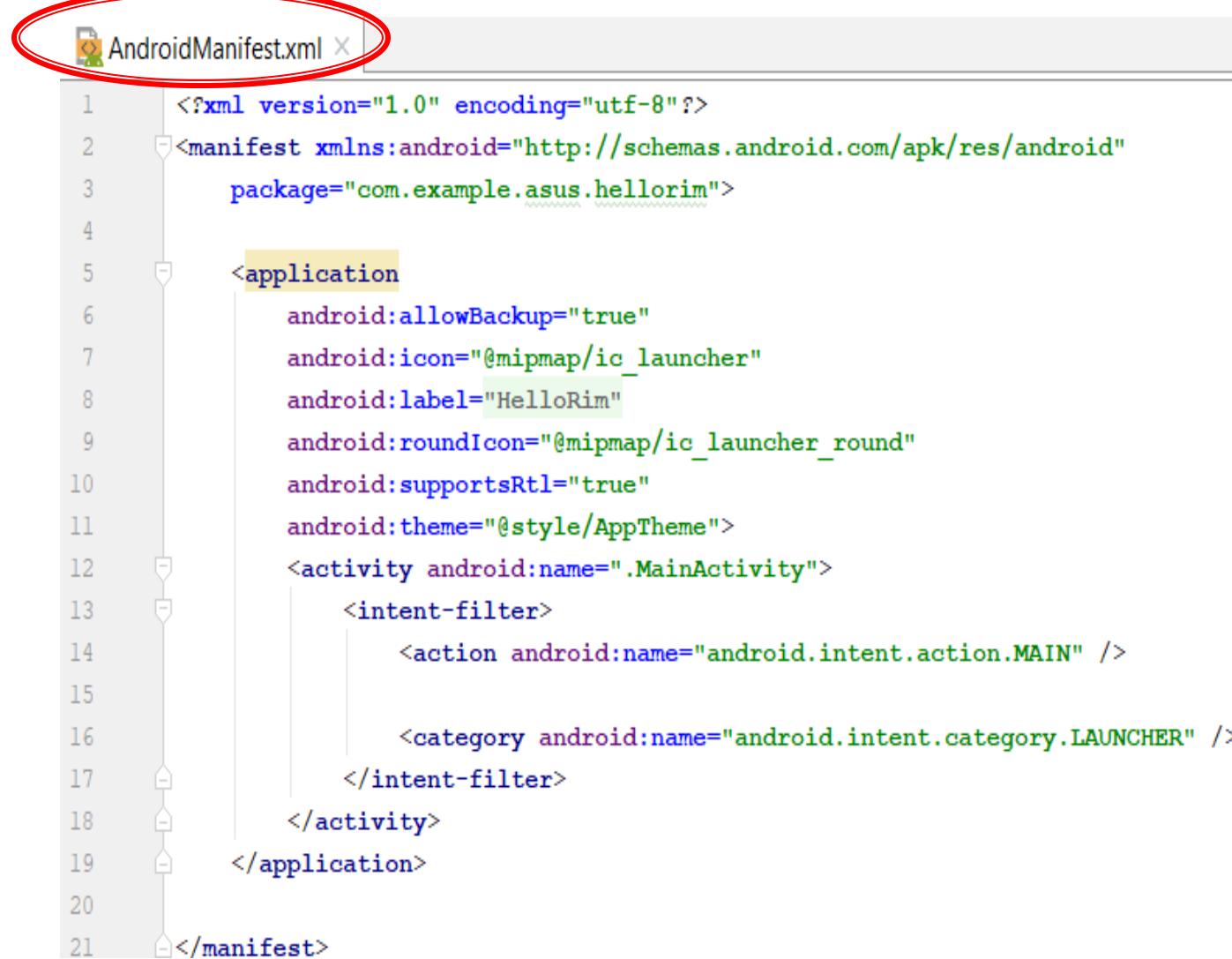
The screenshot shows a code editor window with a red oval highlighting the tab bar where 'MainActivity.java' is selected. The main area displays Java code for a mobile application:

```
1 package com.example.asus.hellorim;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5
6 public class MainActivity extends AppCompatActivity {
7
8     @Override
9     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
```

The code defines a class named `MainActivity` that extends `AppCompatActivity`. It overrides the `onCreate` method to set the content view to `R.layout.activity_main`.

# Première Application Android

## Code source



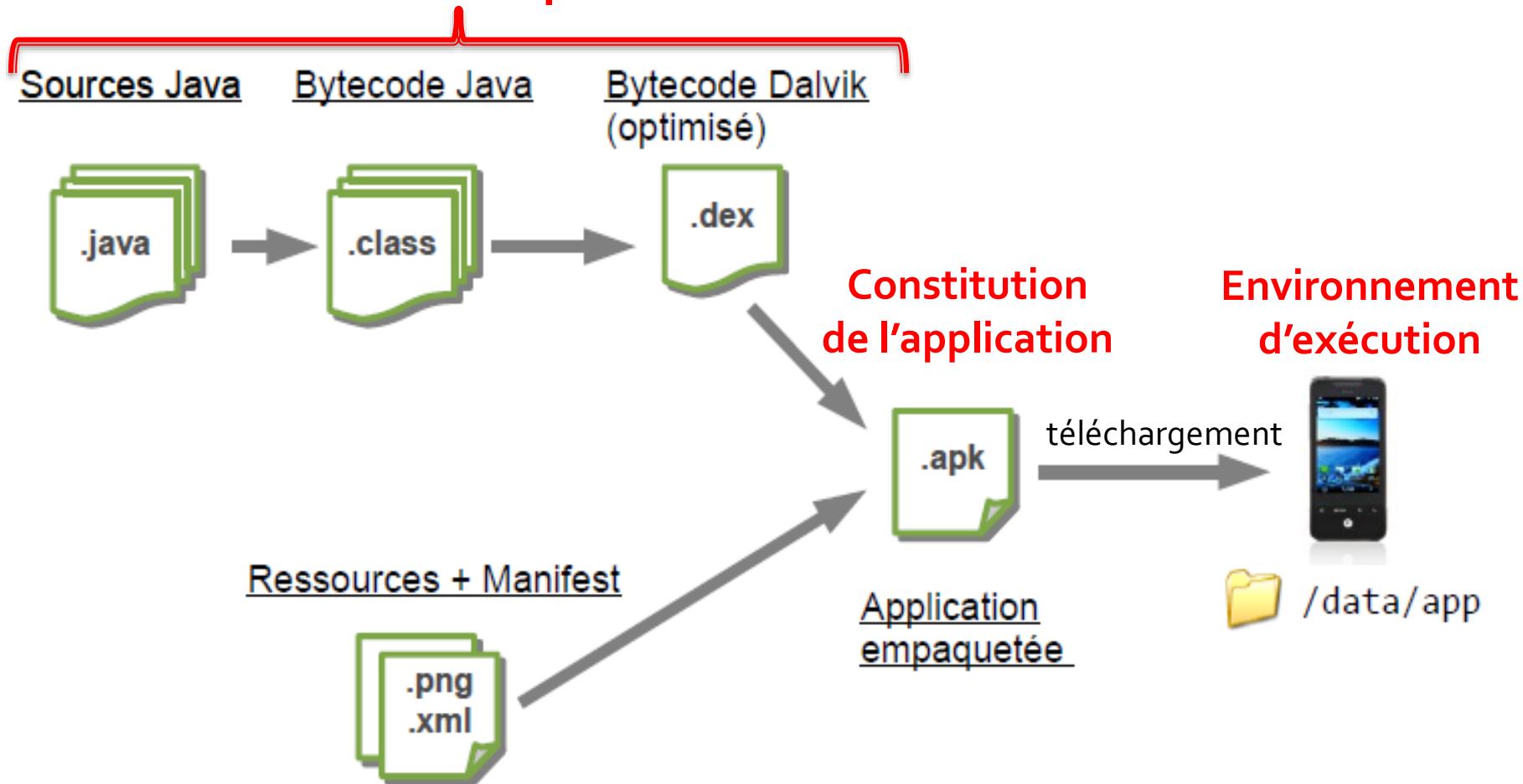
The screenshot shows the code editor of an IDE displaying the `AndroidManifest.xml` file. The file tab at the top left is circled in red. The code itself is an XML manifest for an application named "HelloRim". It defines the application package, its icon, label, theme, and a main activity. The main activity is set to handle both main intent actions and launcher categories.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.asus.hellorim">

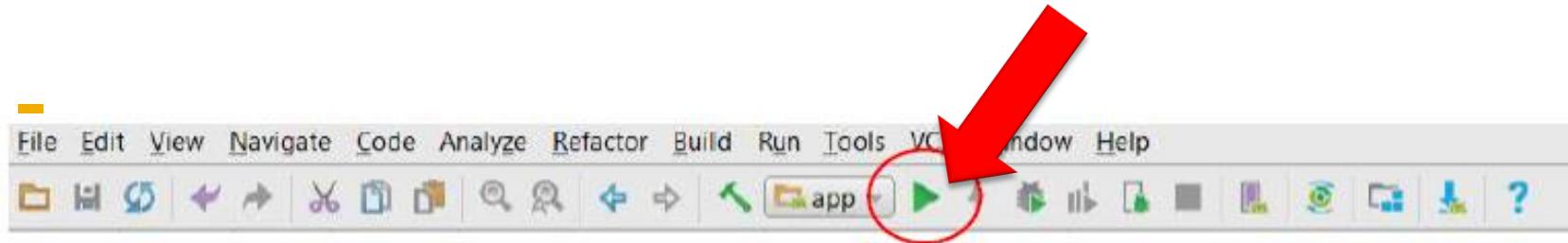
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloRim"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

# Compilation et déploiement d'un projet Android

## Chaine de compilation



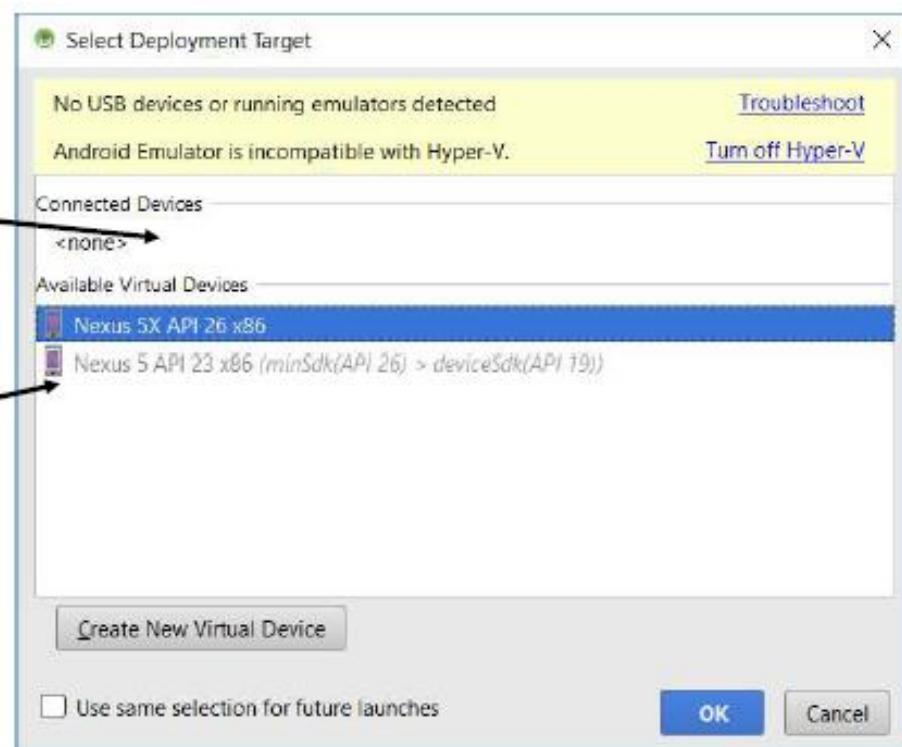
# Exécution de l'application



- Sur un terminal connecté par USB

OU

- Sur un terminal virtuel (simulateur)



# **Chapitre IV**

## **Ressources et Activités d'une Application Android**

# Plan du chapitre IV

- Les ressources
- Utilisation des ressources
- Les activités

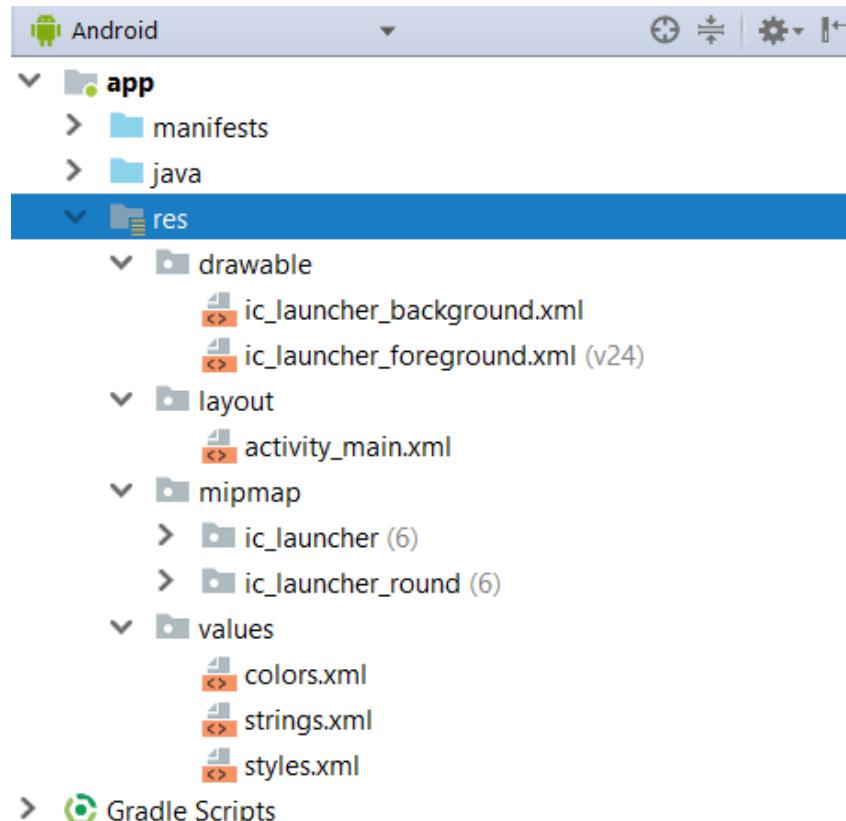
# Les ressources

- Les ressources sont des **fichiers externes** qui sont utilisés par le code et liés à l'application au moment de sa construction.
- Android prévoit l'**externalisation** des ressources.
  - Facilité de maintenance, de mise à jour et de gestion.
- Application embarquée → tout doit être dans le fichier **.apk** téléchargé.

# Les ressources

## Ressources organisées

- Le répertoire **/res** contient toutes les ressources de l'application et comprend quatre sous-répertoires :



# Les ressources

## Ressources organisées

- Le répertoire **/res** contient toutes les ressources de l'application et comprend quatre sous-répertoires :
  - dossier **drawable**, qui contient l'ensemble des images et contenus à afficher à l'écran (image de bouton, un logo, ...)
  - dossier **layout**, qui contient l'ensemble des fichiers layout de votre application (description en XML des interfaces)
  - dossier **mipmap**, qui contient l'icône de l'application
  - dossier **values**, qui contient des définitions en XML de constantes (chaînes, tableaux, couleurs à utiliser dans l'application, ou les styles graphiques à appliquer,...)

# Les ressources

## Création des ressources valeurs

- Les ressources **de type valeur** sont décrites dans des fichiers XML ayant la forme suivante:

```
<?xml version="1.0 " encoding= "utf-8 "?>
<resources>
    <color name= "coulfond">#AA7B03</color>
    <integer name= "limite">567</integer>
    <integer-array name= "codes_postaux">
        <item>64100</item>
        <item>33000</item>
    </integer-array>
    <string name= "mon_titre">Un titre</string>
    <string-array name= "planetes">
        <item>Mercure</item>
        <item>Venus</item>
    </string-array>
    <bool name="actif">true</bool>
    <dimen name "taille">55px</dimen>
</resources>
```

The diagram illustrates the structure of an Android resource XML file. It shows three main components: Type, Nom, and Valeur. Arrows point from these labels to specific parts of the XML code. 'Type' points to the element names like <color>, <integer>, <integer-array>, <string>, <string-array>, <bool>, and <dimen>. 'Nom' points to the <name> attribute within these elements. 'Valeur' points to the values contained within the elements, such as the hex color code #AA7B03, the integer 567, the array items 64100 and 33000, the string 'Un titre', the planet names 'Mercure' and 'Venus', the boolean 'true', and the dimension '55px'.

# Utilisation des ressources

## Référencement d'une ressource

- Référencement d'une ressource dans un fichier xml:

Forme générale :

attribute="@[packageName]:type\_ressource/identificateur\_ressource"

→ Référencement entre ressources: Exemple:

`@string/machaine` fait référence à une chaîne contenue dans un fichier XML placé dans le répertoire **res/values** et définie comme suit:

```
<resources>
    <string name="machaine"> Bonjour </string>
    ...
</resources>
```

- Ex1: `<EditText android:text = "@string/machaine "/>`
- Ex2: `<EditText android:textColor= "@android:color/darker_gray"/>`

# Utilisation des ressources

## La classe R

- Les ressources de l'application sont utilisées dans le code Java à travers la classe statique **R**.
- On obtient une instance de cette classe par **getResources()** de l'activité.
- Principales méthodes de la classe R:
  - boolean **getBoolean(int)**
  - int **getInteger(int)**
  - int[] **getArray(int)**
  - String **getString(int)**
  - String[] **getStringArray(int)**
  - int **getColor(int)**
  - float **getDimension(int)**
  - Drawable **getDrawable(int)**

# Utilisation des ressources

## La classe R

### ■ Référencement d'une ressource dans le code:

Forme générale: **R . typeRessource . identificateurRessource**

- Cette forme est de type **int**.
- Il s'agit de l'identifiant de la ressource.

Exemple : **R.String.machaine**

- On peut récupérer l'instance de la ressource en utilisant cet identifiant et la classe **Resources**.

Exemple :

```
Resources res = getResources();
```

```
String X = res.getString(R.String.machaine);
```

```
Drawable monImage = res.getDrawable(R.drawable.nomImage)
```

# Utilisation des ressources

## La classe R

- Référencement d'une ressource dans le code:

Forme générale: **R . typeRessource . identificateurRessource**

- La méthode **findViewById()** est spécifique pour les **objets graphiques** permettant de les récupérer à partir de leur **id**.

Exemple :

```
TextView texte = (TextView) findViewById(R.id.NomText);  
texte.setText("Hello !");
```

# Utilisation des ressources

## URI: Uniform Resource Identifier

- Désigne l'identifiant uniforme de ressource.
- C'est une **chaine de caractères** identifiant une ressource locale ou distante.
- Sont des URI:
  - les **Uniform Resource Locator (URL)** : identifie une ressource distante sur un réseau et fournit les moyens d'obtenir une représentation de la ressource.

**Syntaxe:** "http://domaine.sous\_domaine/chemin/nom\_du\_fichier"

**Exemple:** http://www.wikipedia.org/ identifie une ressource (page d'accueil Wikipédia) et implique qu'une représentation de cette ressource (page HTML en caractères encodés) peut être obtenue via le protocole HTTP.

- les **Uniform Resource Name (URN)** : identifie une ressource par son nom dans un espace de noms.

**Exemple 1:** urn:isbn:0-395-36341-1 identifie une ressource par un numéro ISBN permettant de référer à un livre.

**Exemple 2:** android.resource://paquetage\_de\_l\_activité/R.type.id\_ressource

# Utilisation des ressources

## Syntaxe d'un URI

- La syntaxe générale d'un URI peut être analysée de la manière suivante:  
**<schéma> : <information> { ? <requête> } { # <fragment> }**  
(les parties entre accolades {} sont optionnelles)
- **Le schéma** décrit la nature de l'information (Exemples: « **tel** » s'il s'agit d'un numéro de téléphone, « **http** » s'il s'agit d'un site internet,...).
- **L'information** est la donnée en tant que telle. Elle respecte une syntaxe qui dépend du schéma. Exemples:
  - « **tel:0606060606** » pour un numéro de téléphone,
  - « **geo:123.456789,-12.345678** » pour des coordonnées GPS,
  - « **http://www.wikipedia.org/** » pour un site internet,...
- **La requête** permet de fournir une précision par rapport à l'information.
- **Le fragment** permet enfin d'accéder à une sous-partie de l'information.

# Utilisation des ressources

## Création d'un URI

- Pour créer un objet URI, c'est simple, il suffit d'utiliser la méthode statique:

**Uri Uri.parse(String uri);**

- Par exemple, pour envoyer un SMS à une personne, on utilisera l'URI :  
**Uri sms = Uri.parse("sms:0606060606");**
- On peut aussi indiquer plusieurs destinataires et un corps pour ce message :  
**Uri.parse("sms:0606060606,0606060607?body=Salut%20lesamies")**

# Les Activités

## Principe

- L'activité est un composant essentiel permettant de gérer l'interactivité avec l'utilisateur **via une interface graphique**.
- Une application est formée de **N** activités
- Pour pouvoir être lancée, une activité **doit être déclarée dans le fichier « Manifest.xml »** de l'application.

# Les Activités

## Etats d'une activité

- Une activité possède 4 états :
  - « **Active** » :  
l'activité est lancée par l'utilisateur, elle **s'exécute au premier plan**.
  - « **En Pause** » :  
l'activité s'exécute, mais elle **a perdu le focus** (n'est plus au premier plan) et elle est partiellement visible.
  - « **Stoppée** » :  
l'activité **n'est plus au premier plan et n'est plus visible**. Elle est complètement recouverte par une autre activité.
  - « **Morte** » :  
L'activité est **détruite**.

# Les Activités

## Méthodes de transition

- Lors des changements d'état, le framework Android appelle les méthodes de transition suivantes:

Méthode	Rôle
<b>onCreate()</b>	Appelée <b>à la création</b> d'une activité. Elle permet de l'initialiser.
<b>onStart()</b>	Appelée quand l'activité est <b>démarrée</b> .
<b>onResume()</b>	Appelée <b>avant que l'activité passe (ou repasse) en premier plan</b> (elle va commencer à interagir avec l'utilisateur)
<b>onPause()</b>	Appelée quand l'activité <b>passe en arrière plan</b> et que le système va démarrer une autre activité.
<b>onStop()</b>	Appelée quand l'activité <b>n'est plus visible</b> .
<b>onRestart()</b>	Appelée quand l'activité <b>redevient visible</b> .
<b>onDestroy()</b>	Appelée quand l'activité est <b>fermée</b> par le système à cause d'un manque de ressources, ou par l'utilisateur à l'utilisation d'un <code>finish()</code> .

# Les Activités

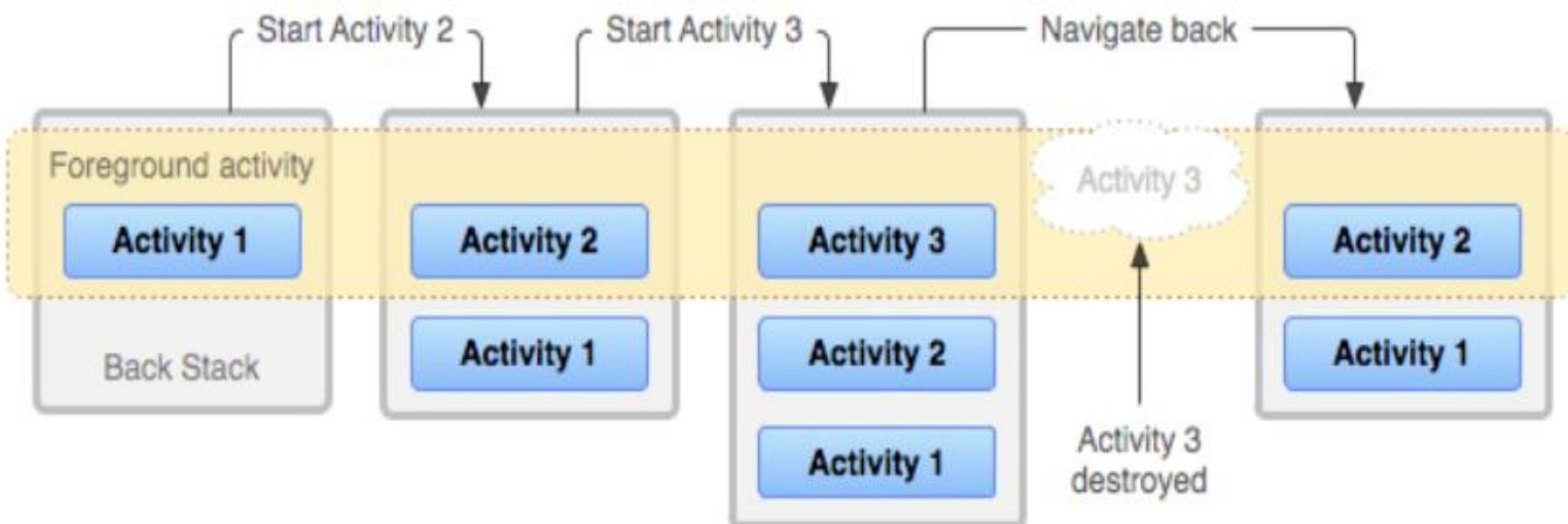
## Fichier activity\_main.xml

- Le fichier activity\_main.xml est repéré par la constante **R.layout.activity\_main**.
- Son contenu peut être visualisé en mode **Design** ou **Text** (XML).
- Ce fichier contient la description de l'IHM qui est souvent construite par glisser-déposer sous l'onglet **Design** qui suggère :
  - les composants graphiques (**Widgets**)
  - les positionnements (**Layouts**)

# Les Activités

## Pile des activités

- Les activités sont cédulées en utilisant la notion de pile.
- La pile est du type « LIFO » (**Last In First Out**).

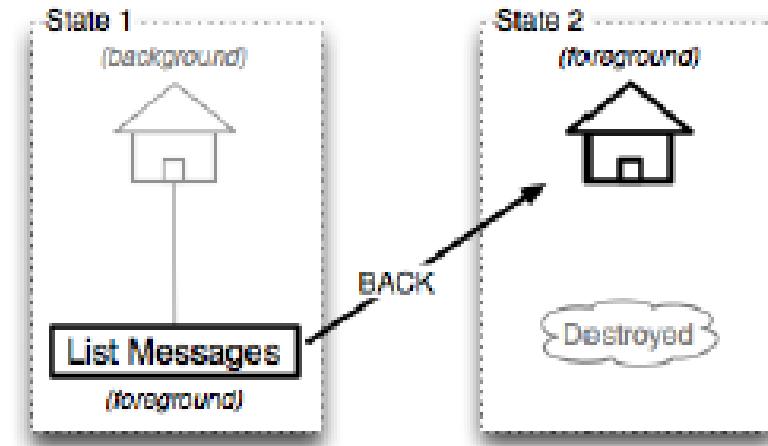


# Les Activités

## Pile des activités

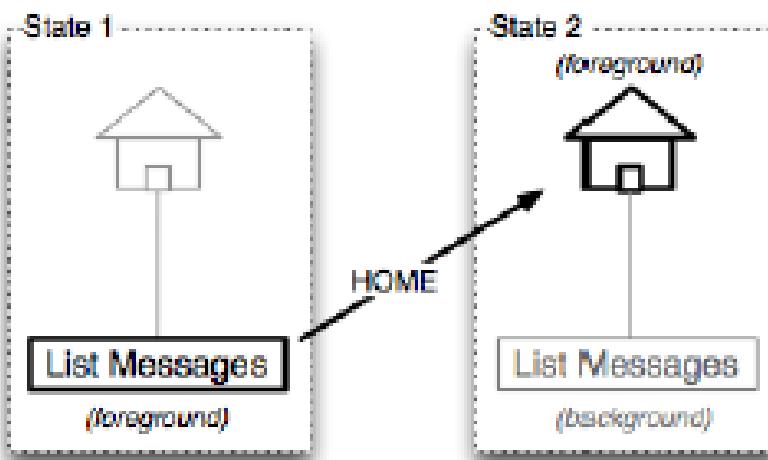
- Les activités sont **empilées/dépilées**

- Empilée quand une activité démarre
- Dépilée (i.e. détruite) quand on presse le **bouton 'BACK'**



- Une pression sur le bouton '**HOME**' ne dépile pas l'activité.

- Elle passe simplement en arrière plan



# **Chapitre VI**

## **Les intentions**

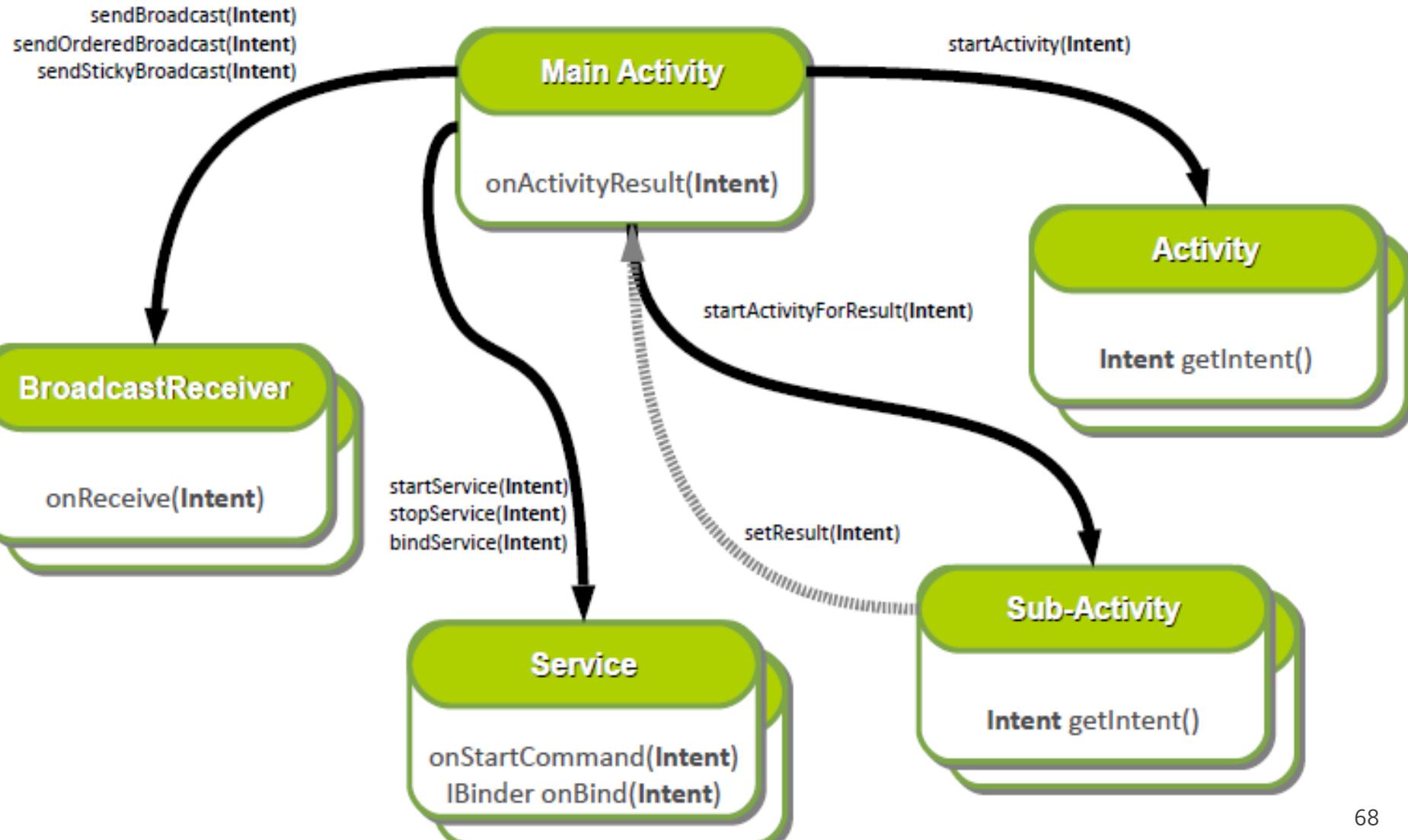
# Plan du chapitre VI

- Principe des intentions
- Fonctionnement des intentions
- Informations d'un intent
- Intent explicite pour une nouvelle activité
- Intent implicite
- Transfert de données entre activités
- Filtres d'intention

# Principe des intentions

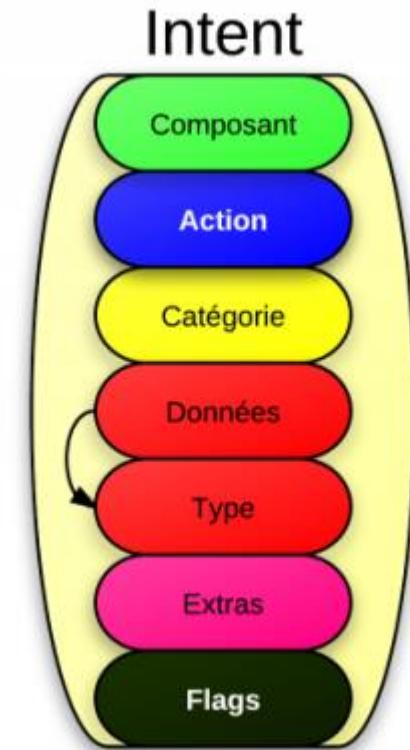
- Une intention est un **message** système qu'on peut qualifier d'**événement**.
- Emis par le terminal pour prévenir les applications de la survenue d'évènements (cas **des évènements systèmes**) ou par toute autre application (cas **des évènements applicatifs**).
  - **Évènements systèmes** : Insertion d'une carte SD, réception d'un SMS, ...
  - **Évènements applicatifs**: Démarrage d'un logiciel, Arrivée d'un utilisateur à Paris en utilisant les informations de géolocalisation du terminal, ...
- Un événement est une sous classe de **android.content.Intent**
- Les Intents permettent de **gérer l'envoi et la réception de messages** afin de faire coopérer les applications.

# Fonctionnement des intentions



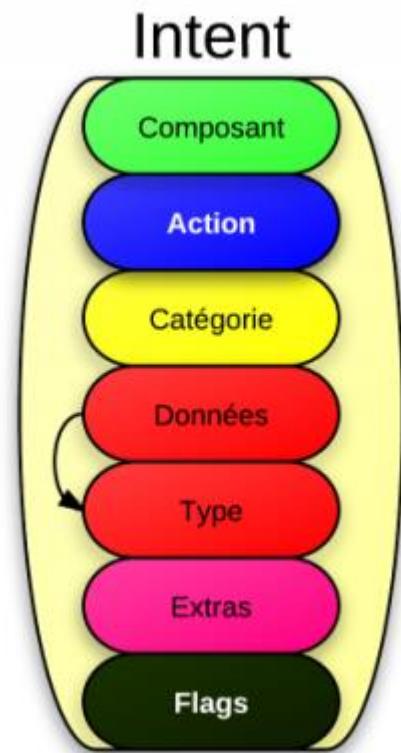
# Informations d'un intent

- Un objet Intent contient plusieurs informations.
- La façon dont sont renseignés ces champs détermine **la nature ainsi que les objectifs de l'intent**.
  - On peut envoyer des Intents informatifs pour faire passer des messages.
  - On peut aussi envoyer des Intents servant à lancer une nouvelle activité.



# Informations d'un intent

- Un objet Intent contient les informations suivantes:
  - **le nom du destinataire (composant ciblé)** (facultatif)
  - **l'action à réaliser** sous forme de chaîne de caractères
  - **la catégorie de l'action**
  - **les données** (contenu MIME et URI) et **leurs types**
  - **des données supplémentaires (extras)** sous forme de paires clé/valeur
  - **des flags**



# Informations d'un intent

## ■ Nom du destinataire

Suivant le renseignement ou pas du nom du destinataire, on distingue deux types d' intents:

### ■ Intent explicite

- le champ du composant ciblé est précisé
- Ce champ est constitué de deux informations : le package où se situe le composant et le nom du composant.
- Ainsi, quand l'intent sera exécuté, Android pourra retrouver le composant de destination de manière précise.

### ■ Intent implicite

- le nom du composant ciblé n'est pas précisé
- C'est pourquoi on va **renseigner d'autres champs** pour laisser Android déterminer qui est capable de réceptionner cet intent.

# Informations d'un intent

## ■ Action d'un intent:

- Ce champ définit **ce qu'on désire que le destinataire fasse** (le traitement à déclencher).
- De nombreuses constantes sont définies dans le SDK pour les actions nativement disponibles comme:
  - **ACTION\_WEB\_SEARCH** pour réaliser une recherche sur Internet
  - **ACTION\_CALL** pour passer un appel téléphonique.
  - **ACTION\_VIEW** pour afficher les données à l'utilisateur.
  - **ACTION\_MAIN** pour commencer en tant que point d'entrée principal.
  - **ACTION\_EDIT** pour fournir un accès éditable aux données fournies.
  - **ACTION\_DIAL** pour composer un numéro comme spécifié par les données. Ceci montre une interface utilisateur avec le numéro composé, permettant à l'utilisateur d'initier l'appel.

## ■ D'autres actions dans :

<https://developer.android.com/reference/android/content/Intent>

# Informations d'un intent

## ■ Les données :

- Ce sont les données sur lesquelles le destinataire doit effectuer son action → Ces données détaillent l'action de l'Intent.
- Pour illustration, dans l'exemple `ACTION_CALL`, la donnée sera le numéro de téléphone à composer.
- Les données peuvent être nulle; certaines actions n'appelant pas forcément des données à devoir être précisées.

### Remarque:

Dans le cas d'un intent implicite, il faut au moins fournir les deux informations essentielles : **action et données**

# Informations d'un intent

Exemples: Voici quelques exemples de paires **action/donnée**:

- **ACTION\_VIEW content://contacts/people/1** - Affiche des informations sur la personne dont l'identifiant est "1".
- **ACTION\_VIEW content://contacts/people/** - Affiche une liste de personnes que l'utilisateur peut parcourir. Cet exemple est une entrée de haut niveau typique dans l'application Contacts, qui affiche la liste des personnes.
- **ACTION\_VIEW geo:49.5000,123.5000** - Ouvrir l'application de géolocalisation à la position donnée (latitude, longitude).
- **ACTION\_DIAL content://contacts/people/1** - Affiche le numéroteur téléphonique avec la personne renseignée.
- **ACTION\_DIAL tel: 123** - Afficher le numéroteur avec le numéro indiqué renseigné.
- **ACTION\_EDIT content://contacts/people/1** - Modifie les informations sur la personne dont l'identifiant est "1".

# Informations d'un intent

## ■ Le champ catégorie:

- Ce champ permet d'apporter des informations supplémentaires sur l'action à exécuter et le type de composant qui devra gérer l'intent.
- La catégorie `Intent.CATEGORY_LAUNCHER` positionne l'activité comme étant exécutable, cette catégorie est utilisée de pair avec l'action `Intent.ACTION_MAIN`.
  - Android positionne les activités de cette catégorie dans le lanceur d'applications.
- La catégorie `CATEGORY_HOME` marque l'activité à afficher au démarrage du téléphone.

# Informations d'un intent

## ■ Le type :

- Ce champ permet d'indiquer quel est le type des données incluses. Normalement ce type est contenu dans les données, mais en précisant cet attribut vous pouvez désactiver cette vérification automatique et imposer un type particulier.

## ■ Les extras :

- Ce champ permet d'ajouter du contenu aux intents afin de les faire circuler entre les composants.

## ■ Les flags (drapeaux) :

- Ces flags permettent de modifier le comportement de l'intent (ex. gérer l'ordre d'une activité dans la pile d'activités).

# Intent explicite pour une nouvelle activité

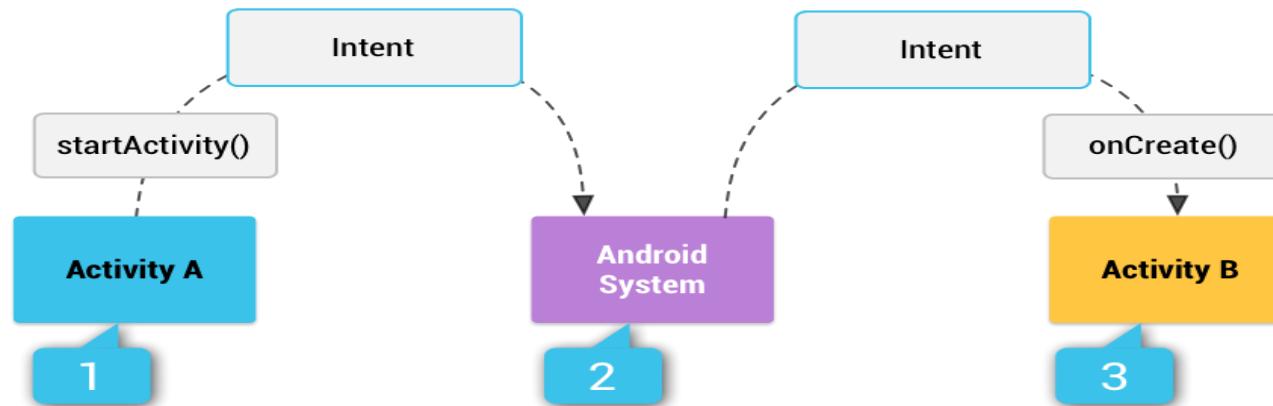
- Pour passer d'une activité courante à une nouvelle activité de l'application, on peut créer l'Intent et passer la classe de l'activité ciblée :

```
Intent i = new Intent(getApplicationContext(), NouvelleActivity.class);  
startActivity(i);
```

- Le premier argument du constructeur de l'Intent doit être le **Context de l'application**.
- NB.: Il faut déclarer la nouvelle activité dans le AndroidManifest.xml.  
`<activity android:name="NouvelleActivity" />`
- La méthode **startActivity** (**Intent intent**) est une méthode de la classe Activity permettant de lancer une autre activité.

# Intent implicite

Comment une intention implicite est livrée par le système Android pour démarrer une autre activité ?



1. L'activité A crée un intent **avec une description de l'action** et le passe pour **startActivity()**.
2. Le système Android cherche toutes les applications ayant un filtre d'intention qui correspond à l'intent.
3. Lorsqu'une correspondance est trouvée, le système démarre l'activité correspondante (activité B) en appelant sa méthode **onCreate()** et en lui transmettant l'intention.

# Intent implicite

## ■ Exemple 1: Consultation des contacts

```
String myData = "content://contacts/people/";  
Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse(myData));  
startActivity(i);
```

## ■ Remarque:

- Pour voir un contact particulier, on peut mettre par exemple:  
"content://contacts/people/**2**"
- Pour éditer un contact, on utilise **Intent.ACTION\_EDIT**

# Intent implicite

## ■ Exemple 2: Consultation d'une page Web

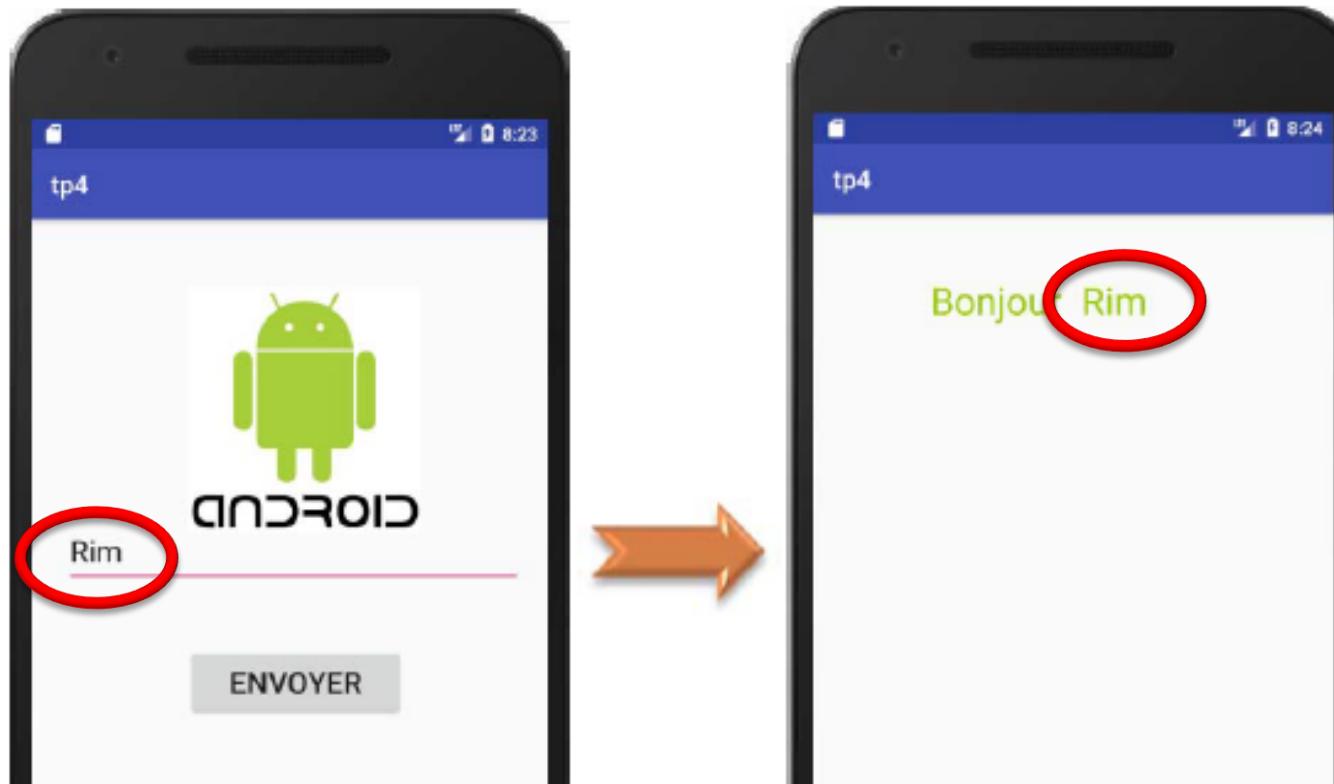
```
String myData = "http://www.youtube.com";  
Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse(myData));  
startActivity(i);
```

- **NB:** Il ne faut pas oublier d'ajouter les permissions nécessaires dans le fichier manifeste :

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

# Transfert de données entre activités

- Les Intent servent parfois **d'enveloppes pour passer des informations** d'une activité à une autre.



# Transfert de données entre activités

- **Le champ extra** d'un intent permet de contenir des données à véhiculer entre les applications.
- Un extra est une clé à laquelle on associe une valeur.
- Pour insérer un extra, il suffit d'utiliser la méthode:  
Intent **putExtra**(String **key**, unType **valeur**)
- Exemple:

```
Intent i = new Intent(getApplicationContext(), Second_Activity.class);
i.putExtra("firstName", nom);
i.putExtra("secondName", prenom);
startActivity(i);
```

# Transfert de données entre activités

- Dans une activité, on récupère l'Intent qui a lancé l'activité par **getIntent()**.
- Pour récupérer tous les extras de l'Intent, on utilise **getExtras()**.
- Pour récupérer un extra associé à une entrée, on utilise :  
**getTypeEntrée(nomEntrée, valeurParDefaut)**
  - valeurParDefaut est la valeur renournée s'il n'y a pas d'extra associé à nomEntrée dans l'Intent.
- Exemple:

```
Bundle extras = getIntent().getExtras();
String X = new String(extras.getString("firstName", null));
ou bien,
String X =getIntent().getExtras().getString(" firstName");
ou bien,
String X =getIntent().getStringExtra(" firstName");
```

# Transfert de données entre activités

- La méthode **startActivity** (**Intent intent**) est une méthode de la classe Activity permettant de lancer une autre activité.
- **Remarque :**
  - Lorsqu'on utilise cette méthode **startActivity**(...), on n'attend pas à ce que la nouvelle activité renvoie un résultat.
  - Si on attend un résultat de retour, alors on utilise la méthode **startActivityForResult**(...).