

## Programmation Mobile - INF4001 -

Nicolas Le Sommer  
Nicolas.Le-Sommer@univ-ubs.fr  
Université de Bretagne Sud

### Plan du cours

1. Un peu d'histoire
2. Développement d'applications mobiles natives vs hybrides
3. Panorama des solutions existantes
4. Développement d'applications mobiles Android
5. Développement d'applications mobiles hybrides Web avec Cordova/Ionic
6. Développement d'applications mobiles hybrides natives avec ReactNative et NatifScript

© UBS/NLS

Master 2 - INF 4001

2

### Un peu d'histoire



Le premier smartphone : IBM Simon Personal Communicator (1994)

### Évolution des terminaux mobiles



Modèle : Psion Revo  
OS : EPOC  
RAM : 8M  
Proc. : ARM 710T à 36MHz  
Date de sortie : 1999  
Comm. : Bluetooth + Serie



Modèle : Ipaq H3630  
OS : WindowsCE  
RAM : 32M  
ROM : 16M  
Proc. : SA-1110 à 206MHz  
Date de sortie : 2001  
Comm. : IrDA



Modèle : Apple MessagePad 100  
OS : Newtown 02  
Date de sortie : 1993



Modèle : Ipaq H5500  
OS : WindowsCE  
RAM : 128M  
ROM : 48M  
Proc. : PXA255 à 400MHz  
Date de sortie : 2003  
Comm. : IrDA, Bluetooth, Wi-Fi



Modèle : Palm TX  
OS : PalmOS  
RAM : 32M  
ROM : 128M  
Proc. : PXA270 à 312MHz  
Date de sortie : 2009  
Comm. : Bluetooth, Wi-Fi, IrDA



Modèle : HTC G1  
OS : Android 1.5, 1.6, 2.1 et 2.2  
RAM : 256Mo  
Stockage: 1Go (32Go carte)  
Proc. : Qualcomm MSM7201A à 528 MHz  
Date de sortie : 2008  
Comm. : Bluetooth, Wi-Fi, GSM, 3G

Modèle : Nokia N9  
OS : MeeGo  
RAM : 1Go  
Stockage: 64Go  
Proc. : ARM Cortex A8 1GHz  
GPU : SGX530  
Date de sortie : 2010  
Comm. : Bluetooth, Wi-Fi GSM, 3G



## Évolution des terminaux mobiles



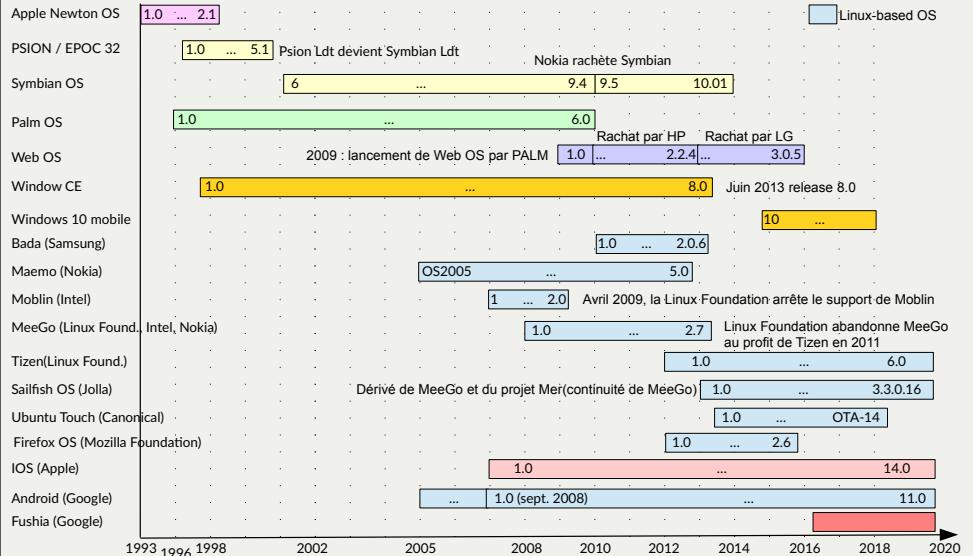
Modèle : Samsung S20  
OS : Android 10  
RAM : 8Go  
Mémoire interne : 128Go  
Proc. : Octo-Core à 2.73GHz, 2.5GHz, 2Ghz  
GPU : Mali-G77 MP11  
Date de sortie : 2020  
Taille d'écran : 6.2 pouces (3200 x 1440 pixels)  
Comm. : Wi-Fi, Bluetooth, LTE, ...



Modèle : OnePlus 8T  
OS : Android 9.0  
RAM : 8Go/12Go  
Mémoire interne : 128Go  
Proc. : 8 coeurs (4 à 2,1GHz et 4 à 2GHz)  
GPU : ARM Mali-G72 900MHz  
Date de sortie : 2020  
Taille d'écran : 6,4 pouces AMOLED (2400 x 1080 pixels)  
Comm. : Wi-Fi, Bluetooth, LTE, 5G ...

Modèle : Planetcom Cosmo Communicator  
OS : Android 9.0 / Linux  
RAM : 6Go  
Mémoire interne : 128Go  
Proc. : 8 coeurs (4 à 2,1GHz et 4 à 2GHz)  
GPU : ARM Mali-G72 900MHz  
Date de sortie : 2018  
Taille d'écran : 5,9 pouces (2160x1080 pixels)  
Comm. : Wi-Fi, Bluetooth, LTE, ...

## Systèmes mobiles : des réussites et des flops...

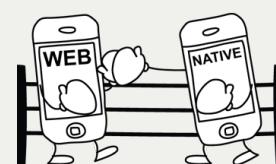


## Parts de marché des systèmes

OS	Parts de marché en %
Android	75
iOS	24,7
Windows	0,2
Autre	0,1

Chiffres 2020 donnés par Kantar World Panel

## Développement d'applications mobiles natives vs hybrides



# Les différentes approches de mise en œuvre

## 1) Sites Web adaptés aux terminaux mobiles

- Développement Web responsive (HTML, CSS, Javascript, PHP, ...)
- Accessible depuis un serveur Web
  - connectivité (quasi) permanente
- Cas particulier : les applications Web progressives

## 2) Applications natives

- Développées en utilisant les API propres à chaque système mobile

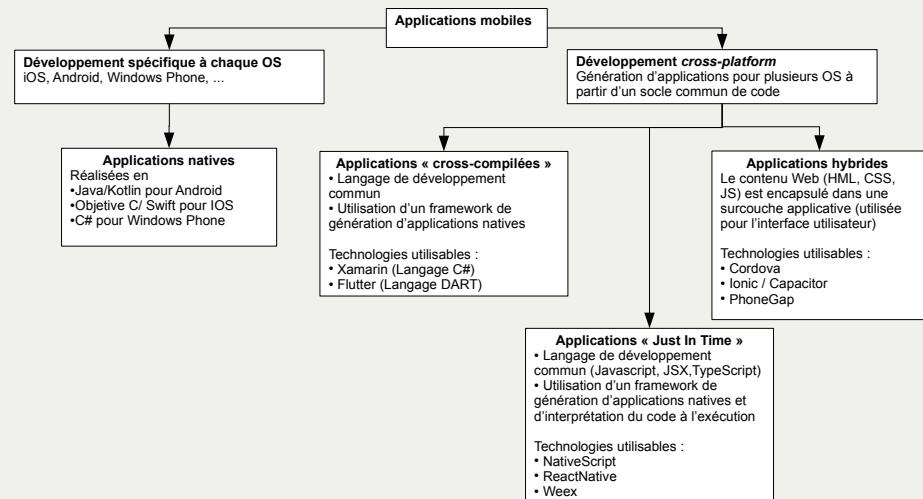
## 3) Applications hybrides multi-plateformes

## 4) Applications multi-plateformes « cross-compiled »

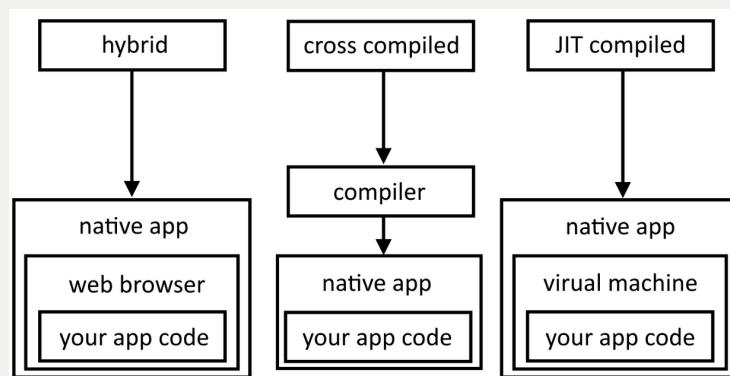
## 5) Applications multi-plateformes de type « Just In Time »

- Les applications de types 2, 3, 4 et 5 peuvent être installées depuis un marché d'applications

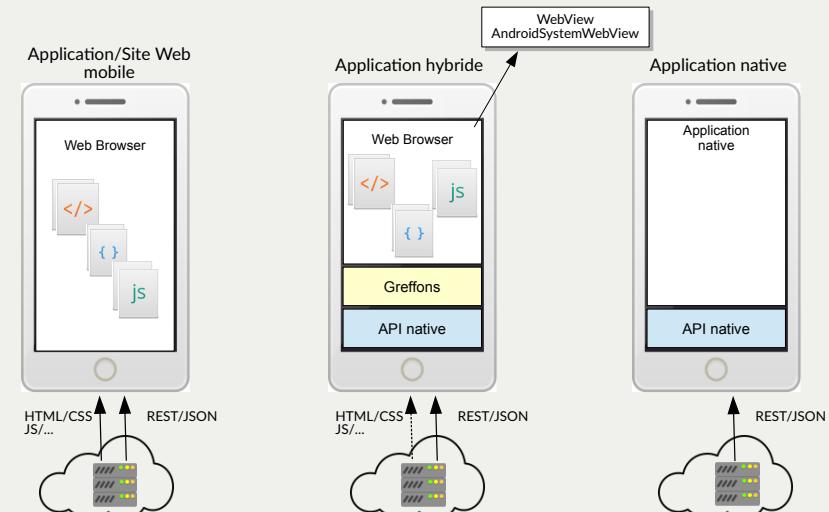
# Les différentes approches de mise en œuvre



# Hybride vs Cross-Compiled vs JIT Compiled



# Principe de fonctionnement



## Les applications Web progressives

- Pour être qualifiées de progressives, une application Web doit être :
  - **découvrable**, afin que le contenu soit trouvé à l'aide de moteurs de recherche ;
  - **installable**, afin d'être disponible sur l'écran d'accueil de l'appareil ;
  - « **liable** », afin qu'elle puisse être partagée simplement via une URL ;
  - « **autonome** », pour qu'elle fonctionne hors-ligne ;
  - **progressive**, pour qu'elle soit utilisable sur les plus vieux navigateurs, mais aussi sur les plus récents ;
  - « **engageante** », afin qu'elle soit capable d'envoyer des notifications lorsque du nouveau contenu est disponible.
  - « **responsive** », afin qu'elle fonctionne sur n'importe quel dispositif (téléphones mobiles, tablettes, ordinateurs portables, ...)
  - **sûre**, afin que la connexion entre l'application et le serveur soit sécurisée.

## Les applications Web progressives

- Les applications Web progressives reposent sur :
  - Les ServiceWorkers
    - se comportent comme des serveurs proxy entre les applications Web, le navigateur et le réseau
    - prennent la forme de code Javascript qui contrôle une page (la page qui lui est associée) afin d'intercepter et de modifier les requêtes de navigation et d'accès aux ressources, et de mettre en cache des données.
      - La mise en cache de données permet
        - un démarrage plus rapide de l'application
        - un fonctionnement hors-ligne
  - D'autres fonctionnalités des navigateurs telles que
    - *Web App Manifest*,
    - *Push*,
    - *Notifications*,
    - et *Add to Home Screen*

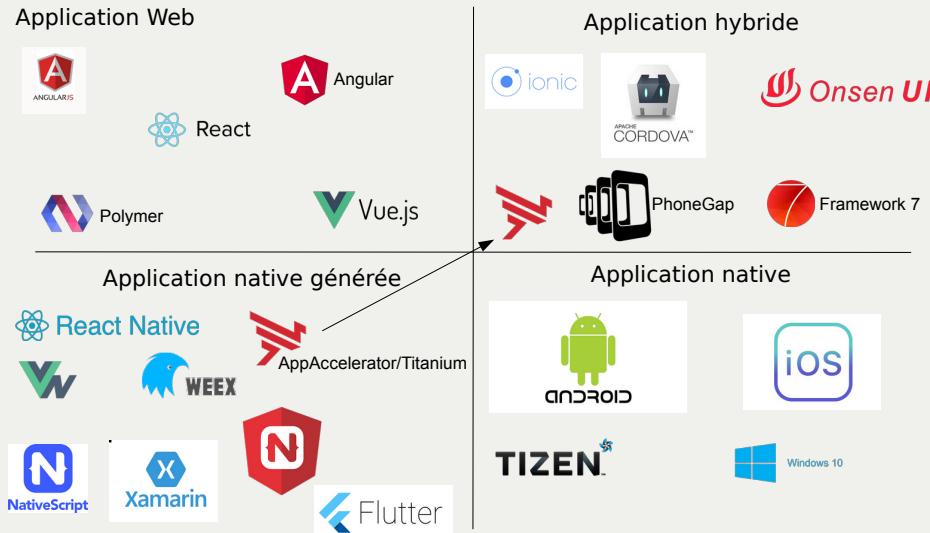
[https://developer.mozilla.org/en-US/docs/Web/Progressive\\_web\\_apps](https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps)

## Quelle approche choisir ?

	Application Web	Application hybride	Application native
Coût de développement	+	++	+++
Rapidité de déploiement / mise à jour	+++	++	+
Mode hors ligne	-	+	+
Performances (rapidité/fluidité)	+	++	+++
Accès aux fonctions natives du terminal mobile (GPS, capteurs, notifications, ...)	-	+	+++

## Panorama des solutions existantes

## Panorama des solutions existantes

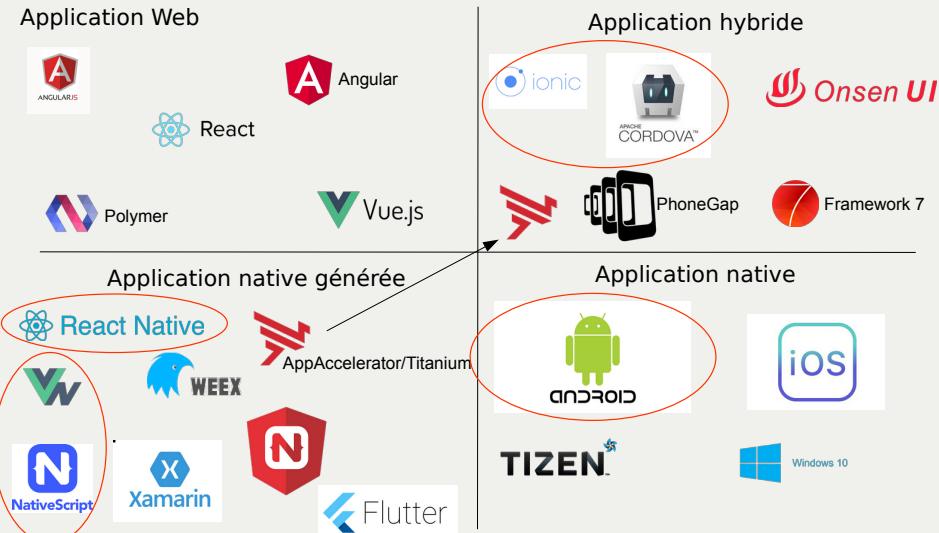


© UBS/NLS

Master 2 - INF 4001

17

## Panorama des solutions existantes



© UBS/NLS

Master 2 - INF 4001

18



## Programmation d'applications mobiles Android - INF4011 -

Nicolas Le Sommer

Nicolas.Le-Sommer@univ-ubs.fr

Université de Bretagne Sud, IRISA

## Plan du cours

1. La plateforme Android
2. Les outils de développement
3. Structure d'une application
4. Les principaux concepts
5. Les interfaces graphiques
6. Programmation concurrente

© UBS/NLS

Master 2 - INF 4001

2

## Quelques chiffres clés

- 2005 : Android est créé par la société éponyme
  - Société rachetée par Google
  - Plateforme conçue pour un appareil photo
- 2007 : lancement de la première version d'Android par Google
- 11 : c'est nombre de versions d'Android
- Android équipe une large majorité des téléphones/tablettes vendus(es) dans le monde

## La plateforme Android

## La plateforme Android

- Android est une plateforme qui
  - est basée sur le système d'exploitation Linux
  - offre un environnement d'exécution pour des applications développées en Java
    - ART : *Android Runtime* (remplace la JVM Dalvik depuis Android 5)
  - dispose d'un environnement de développement
    - API/SDK/NDK, Android Studio, AVD (*Android Virtual Device*)
  - est adaptée à différents équipements
    - Tablettes et téléphones, montres connectées (*Android Wear*), télévision (*Android TV*), automobile (*Android Auto*), les ordinateurs (*Android-x86*), les objets connectés (*Android Things*)
  - est open source
    - *Android AOSP* (<https://source.android.com>)



## Écosystème Android

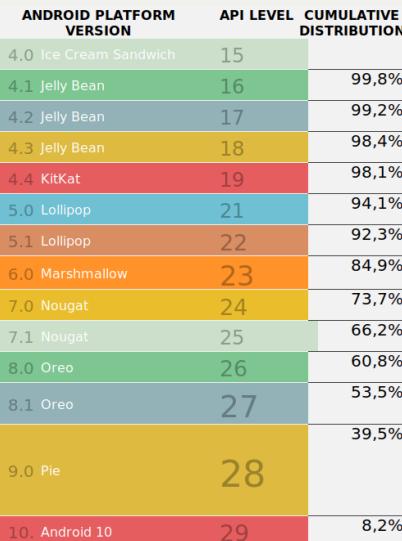
- Applications Google
  - Services *Google Play*, *GMail*, *Maps*, *PlayStore*, *Messages*, *Chrome*, *Youtube*, Moteur de recherche Google, ..
- Services
  - Service de localisation
  - ...
- Magasin d'applications
  - *PlayStore* et ses déclinaisons
    - *Play Films*, *Play Jeux*, *Play Livres*, ...
  - Alternative : F-Droid

## Les versions d'Android

- De nombreuses versions depuis la sortie de la version 1 en 2007

Version	Nom de code	Date de sortie	Version API
2.2.X	Froyo	Mai 2009	9
2.3.X	Gingerbread	Dec. 2010	10
3.X.X	HoneyComb	Fév. 2011	11, 12, 13
4.0.X	Ice Cream	Oct. 2011	14, 15
4.1.X, 4.2.X, 4.3.X	Jelly Bean	Juil. 2012- Juil. 2013	16, 17, 18
4.4.X	KitKat	Oct. 2013	19
5.0.X	Lollipop	Nov. 2014	21
5.1.X	Lollipop	Mars. 2015	22
6.0	Marshmallow	Oct. 2015	23
7.0.x	Nougat	Août. 2016	24
7.1	Nougat	Oct. 2016	25
8.0.X	Oreo	Août 2017	26
8.1.X	Oreo	Dec. 2017	27
9.0	Pie	Août 2018	28
10.0		Sept. 2019	29
11.0		Sept. 2020	30

## Les versions d'Android



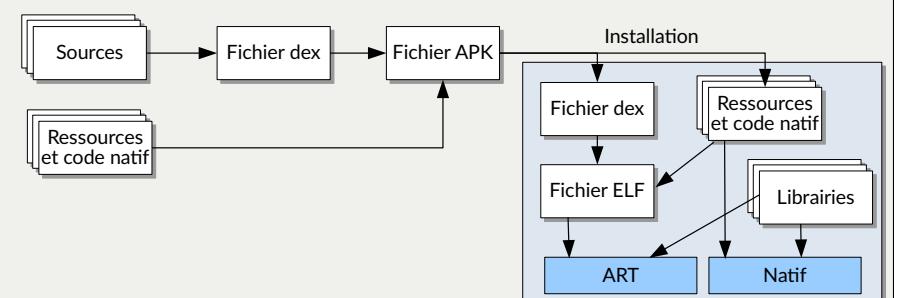
## Architecture de la plateforme Android



## Android Runtime

- Android Runtime

- remplace la machine virtuelle Dalvik depuis la version 5 d'Android
- traduit le code Java en code natif
  - Gain en performance et en autonomie des batteries de 20 % à 30 %
  - Les applications prennent plus de place (+ 20%)



## Exécution des applications

- Chaque application s'exécute dans son propre bac à sable (*sandbox*)
  - Chaque application est considérée comme un « utilisateur différent »
    - Linux est un système d'exploitation multi-utilisateurs
    - Android affecte un identifiant d'utilisateur unique à chaque application
  - Les droits d'accès aux fichiers d'une application sont restreints à celle-ci (restreints à l'utilisateur qu'elle représente)
  - Chaque processus à sa propre machine virtuelle
    - Le code d'une application est isolé de celui d'une autre
    - Chaque application à son propre processus
  - Chaque application ne peut accéder qu'aux composants dont elle a besoin pour s'exécuter (contacts utilisateur, caméra, ...)
    - Les permissions d'accès doivent être explicitées
    - Principe du moindre privilège

## Exécution des applications

- Pour libérer des ressources (mémoire, processeur), Android peut arrêter une application (et tuer le processus associé)
- Android va sélectionner en priorité les applications qui sont démarrées mais qui sont les moins utilisées
  - Les applications qui ne sont pas affichées à l'écran

## Outils de développement

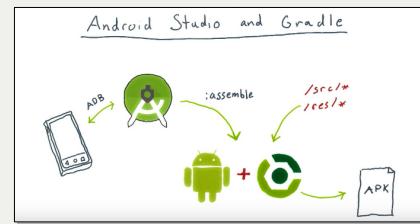
## Java, Kotlin, SDK, NDK, Gradle, CMake, ...

- Une application Android peut être développée dans les langages de programmation Java ou Kotlin
  - Kotlin pour Android est interopérable avec Java (API Android)
    - Programmation orientée objet et fonctionnelle, typage statique
    - Compilation vers Java et Javascript
    - <https://kotlinlang.org/>
  - Elle peut intégrer du code natif à travers une librairie (.so) écrite en C/C++
    - Le code Java appelle le code natif à travers JNI (*Java Native Interface*)
    - Les librairies natives sont compilées à l'aide de l'outil Cmake
    - NDK



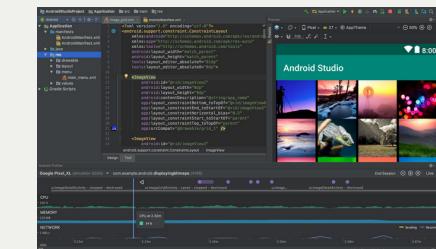
## Java, Kotlin, SDK, NDK, Gradle, CMake, ...

- L'outil Gradle est utilisé pour
  - gérer les dépendances
  - compiler le code source de l'application
  - créer l'APK
  - déployer l'APK sur des terminaux
- Gradle permet de construire une application de manière modulaire
  - *settings.gradle*
    - Modules du projet
  - *build.gradle*
    - Configuration du projet

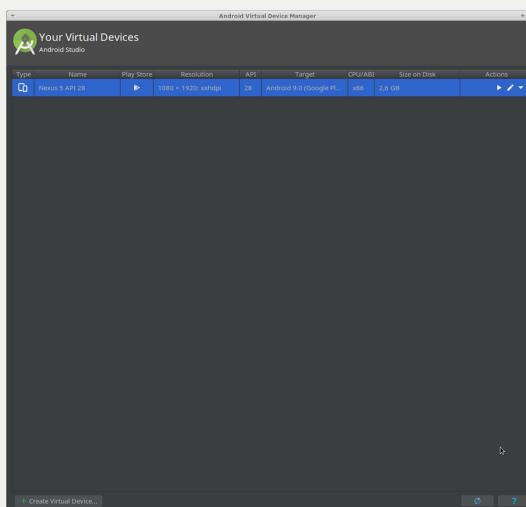


## Android Studio

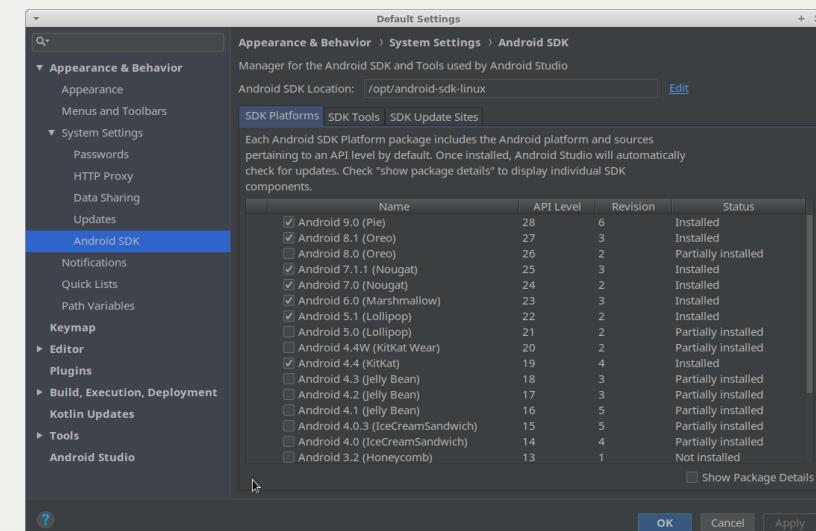
- Environnement de développement
  - reposant sur IntelliJ IDEA de JetBrains
  - permettant le développement d'applications Android en Java, Kotlin, C/C++
  - intégrant différents outils pour faciliter le développement
    - Éditeur graphique d'interfaces utilisateur (layout)
    - Analyseur d'APK
    - Émulateurs de terminaux mobiles
    - Profiler de code



## AVD : Android Virtual Device



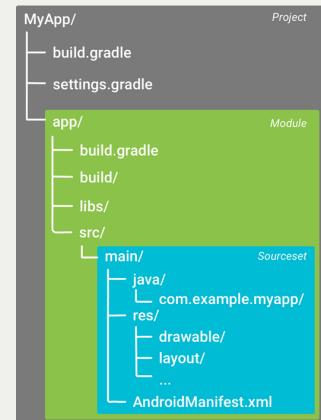
## Gestion des plateformes dans Android Studio



## Structure d'une application

## Structure d'une application

- Une application Android respecte une structure de projet Gradle
- Une application peut contenir
  - un fichier AndroidManifest.xml
  - du code Java/Kotlin
  - du code natif
  - des ressources
    - Interfaces utilisateurs (*layouts*)
    - Images
    - Fichiers XML décrivant
      - le style de l'application
      - les couleurs de l'application
      - des chaînes de caractères adaptées aux paramètres linguistiques du terminal



## Le fichier *AndroidManifest.xml*

- Fichier précisant
  - le nom du paquetage Java de l'application
  - les composants de l'application
    - Activités, services, fournisseurs/gestionnaires de contenus(*Content Providers*), receveurs de messages (*Broadcast Receivers*)
  - les permissions
  - les pré-requis matériels et logiciels de l'application
    - Librairies logicielles, compatibilité d'écran, version(s) de SDK, capteurs utilisés, interfaces de communications requises, ...
  - Des filtres sur les *intents*
    - *Intent* : message de communication de haut niveau défini dans l'API Java permettant à un composant (activité, service, etc) de demander à un autre composant de réaliser une action, avec éventuellement les données communiquées

## Exemple de fichier *AndroidManifest.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ubs.univ.fr.contactbackup">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        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>
        <service android:label="@string/app_name" android:name=".SyncService" />
    </application>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-sdk android:minSdkVersion="19" android:targetSdkVersion="27" />
</manifest>
```

paquetage

Activité principale démarrée au lancement de l'application

Service

Permissions

Comptabilité  
SDK

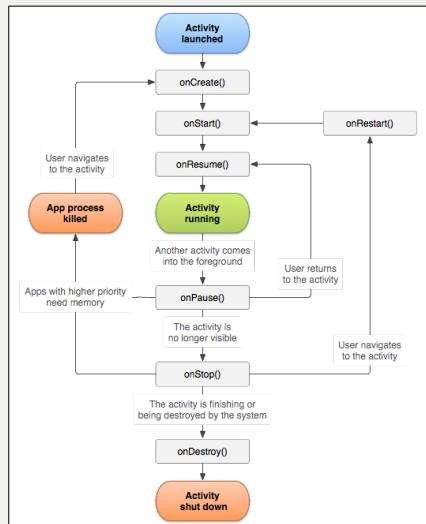
Les ressources commencent par @

## Les activités

- Une activité étend la classe `android.app.Activity`
- La plupart des activités permettent de créer une fenêtre et d'afficher une vue à l'utilisateur ou un fragment de vue (`android.app.FragmentActivity/android.app.Fragment`)
  - Fonction `setContentView()`
- Une activité permet à un utilisateur de réaliser une action
- Une activité doit être déclarée dans le fichier `AndroidManifest.xml`

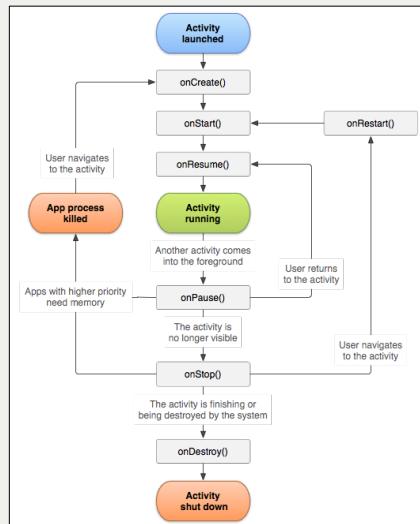
## Les principaux concepts

## Cycle de vie d'une activité



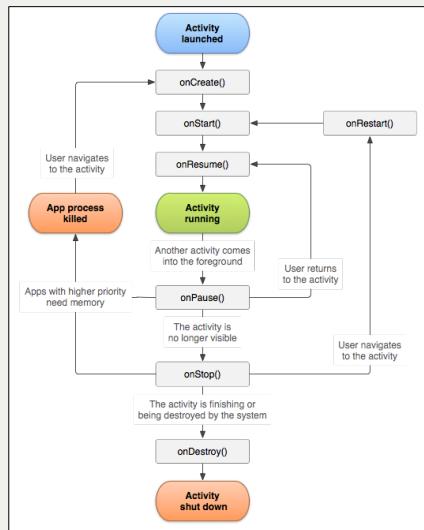
- `onCreate()` :
  - Méthode exécutée à la création de l'activité
  - Si c'est la première activité d'une application, cette méthode est la première invoquée lors du démarrage de l'application
  - Méthode utilisée pour initialiser :
    - la vue XML (`layout`)
    - si nécessaire, les fichiers/données temporaires

## Cycle de vie d'une activité



- `onRestart()` :
  - Méthode exécutée lors du redémarrage de l'activité après un arrêt
    - Arrêt provoqué par un appel à la méthode `stop()`.
  - Cette méthode est donc appelée quand l'application repasse au premier plan après un arrêt prolongé

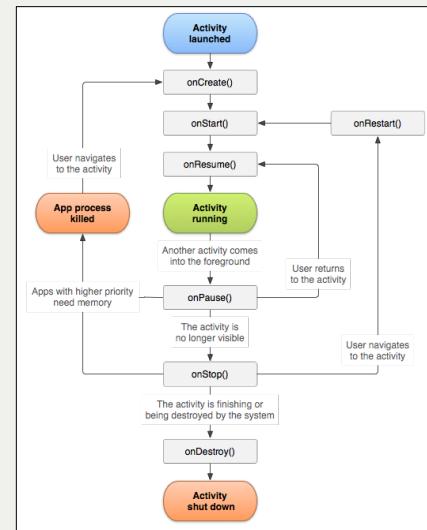
# Cycle de vie d'une activité



- **onStart()** :

- Méthode exécutée après chaque `onCreate()` ou `onRestart()`
- Peut être utilisée pour recharger si nécessaire des données sauvegardées lors du dernier arrêt

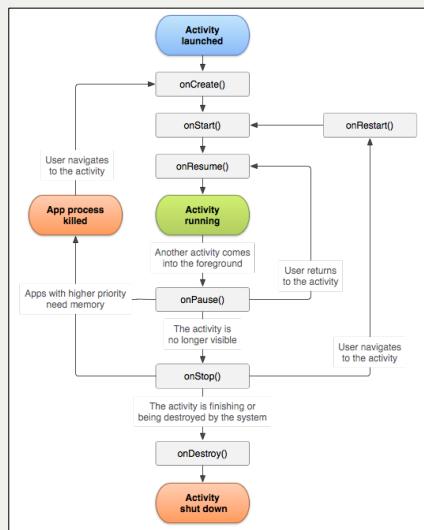
# Cycle de vie d'une activité



- **onResume()** :

- Méthode exécutée après chaque `onStart()`
- Méthode exécutée à chaque passage en premier plan de l'activité (si pas de `stop()`)
- Si nécessaire :
  - gérer la connexion à la base de données
  - mettre à jour des données qui auraient pu être modifiées entre temps (avant le `onResume()`)

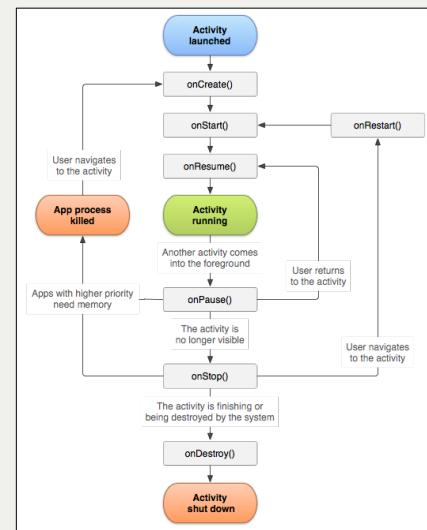
# Cycle de vie d'une activité



- **onPause()** :

- Méthode exécutée à chaque fois que
  - l'utilisateur passe à une autre activité
  - la méthode `finish()` est invoquée sur l'activité
  - Le système a besoin de libérer de la mémoire
- Méthode invoquée avant chaque `onStop()`
- Si nécessaire sauvegarder les données avant qu'elles ne soient perdues après l'arrêt de l'application
- Attention la méthode doit s'exécuter rapidement pour que l'activité suivante puisse démarrer

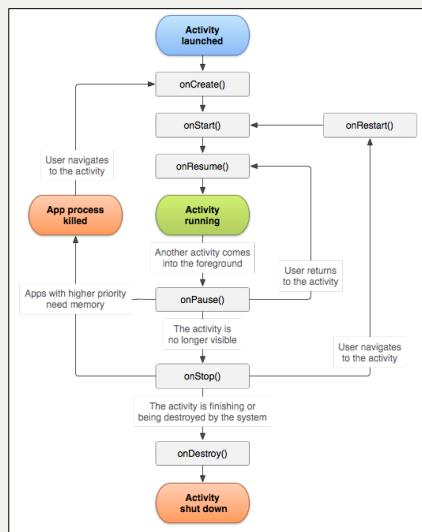
# Cycle de vie d'une activité



- **onStop()** :

- Méthode exécutée avant chaque mise en sommeil
- Méthode exécutée avant la méthode `onDestroy()`
- Libération des ressources

## Cycle de vie d'une activité



- `onDestroy()` :

- Méthode exécutée lors de l'arrêt de l'activité
- La méthode `onCreate()` devra être à nouveau appelée pour recréer l'activité
- Si nécessaire, libérer les fichiers temporaires.

## Exemple d'activité

```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
  
    @Override  
    protected void onStart() {  
        readConfigurationParameters();  
        createTemporaryDataDirectory();  
    }  
  
    @Override  
    protected void onPause() {  
        writeTemporaryData();  
    }  
  
    @Override  
    protected void onResume() {  
        readTemporaryData();  
    }  
  
    @Override  
    protected void onStop() {  
        storeConfigurationParameters();  
    }  
  
    @Override  
    protected void onDestroy() {  
        destroyTemporaryFiles();  
    }  
}
```

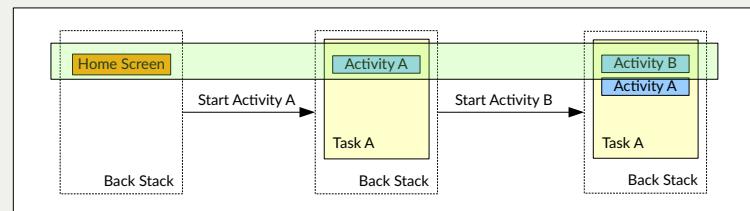
Affichage de la vue définie par le layout `activity_main`  
Fichier `/res/layouts/activity_main.xml`

## Gestion de la pile d'activités

- Une application Android est généralement composée de plusieurs activités
- Les nouvelles activités sont démarrées via des *Intents*
- Les anciennes activités sont détruites ou placées dans une pile appelée « *back stack* »
- Les nouvelles activités peuvent être démarrées dans la même tâche ou dans une nouvelle tâche
- La « *back stack* » est une structure de données qui conserve les activités en arrière plan pour permettre à l'utilisateur d'y revenir (naviguer) avec le bouton *back*
- La « *back stack* » fonctionne dans un mode LIFO (Last In First Out)
- Elle peut être effacée en cas de faible mémoire

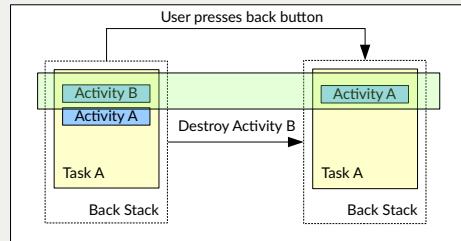
## Tâches

- Une tâche est une collection d'activités créées lors de l'utilisation d'une application
- Une activité est lancée dans une nouvelle tâche depuis l'écran Home
- Par défaut toutes les activités d'une application appartiennent à une même tâche
- Les activités dans une tâche peuvent être en avant plan, en arrière plan ou dans la « *back stack* »



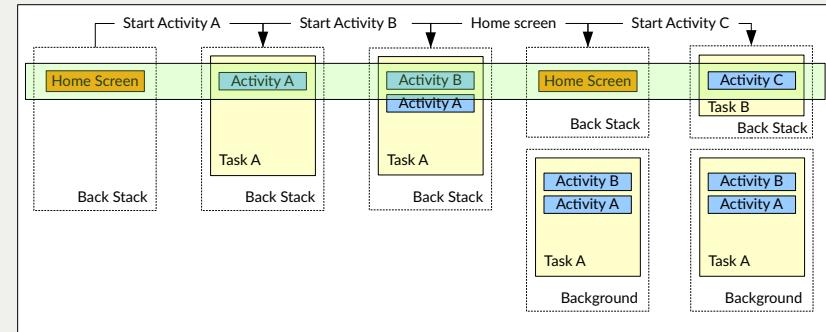
## Retour d'une activité au premier plan

- Lorsque l'utilisateur presse la touche retour, l'activité
  - qui est au premier plan est dépilerée et détruite
  - suivante dans la pile repasse au premier plan



## Tâches en arrière plan

- Lorsqu'une nouvelle application est démarrée, l'application qui était au premier plan passe en arrière plan, et la nouvelle prend sa place dans la « *back stack* »



## Modification des directives de lancement d'une activité

- Les activités peuvent être paramétrées avec les modes de lancement suivants dans le fichier *AndroidManifest.xml*
  - *standard*
    - Mode de lancement par défaut
    - Toutes les activités appartiennent à la même tâche dans une application
  - *singleTop*
    - Une seule instance d'une activité peut exister en haut de la « *back stack* »
    - Plusieurs instances peuvent exister s'il y a des activités entre elles
  - *singleTask*
    - L'activité est à l'origine de la tâche (activité « racine »)
    - Les autres activités font partie de la tâche si celles-ci sont lancées par l'activité « racine »
    - Si l'activité existe déjà, les *intents* sont adressés à cette instance
  - *singleInstance*
    - L'activité qui est à l'origine de la tâche est la seule de sa tâche
    - Ce mode peut aussi être défini dans un *intent* via la méthode *addFlags()*

## Les services Android

- Un service
  - est un composant d'application qui peut tourner en arrière plan/tâche de fond
  - n'a pas d'interface contrairement à une activité
  - s'exécute dans le même processus que celui de son application
  - n'est pas un thread
- Il peut être démarré
  - par un autre composant d'une application via la méthode *startService()*
    - ce service peut continuer à s'exécuter même si l'utilisateur bascule sur une autre application
  - lorsqu'un composant se lie au service via la méthode *bindService()* pour interagir avec ce dernier, voire faire de la communication inter-processus (IPC : interprocess communication)

## Les services Android

- Lorsqu'un service est démarré son cycle de vie est indépendant de celui du composant qui l'a démarré
- Un service peut être arrêté
  - par un autre composant d'une application en invoquant la méthode `stopService()`
  - par lui-même en invoquant la méthode `stopSelf()`
  - par le système s'il n'y a plus de composants qui lui sont liés.
- Un service doit implémenter la classe `Service`.
- Un service doit être déclaré dans le fichier `AndroidManifest.xml`
  - `service` est un élément fils de l'élément `application`

```
<manifest ... >
  ...
  <application ... >
    <service android:name=".ExampleService" />
  ...
</application>
</manifest>
```

Nom de la classe  
Ici, le nom du paquetage est le même que celui de l'application

## Les types de services

- 3 types de services :
  - **Foreground**
    - Service réalisant une opération qui peut être notifiée à l'utilisateur
      - Ils doivent afficher une notification
    - Continuent de s'exécuter même si l'utilisateur n'interagit pas avec l'application
  - **Background**
    - Réalise une opération qui n'est pas directement notifiée à l'utilisateur
  - **Bound**
    - Un service est lié lorsqu'un composant d'une application invoque la méthode `bindService()`.
    - Un « service lié » offre une interface client/serveur qui permet à un composant d'envoyer une requête et de recevoir une réponse, et cela potentiellement à travers IPC
    - S'exécute tant qu'un composant est lié au service
      - Plusieurs composants peuvent se lier au service une seule fois. Quand, il n'y a plus de composants liés, le service est détruit

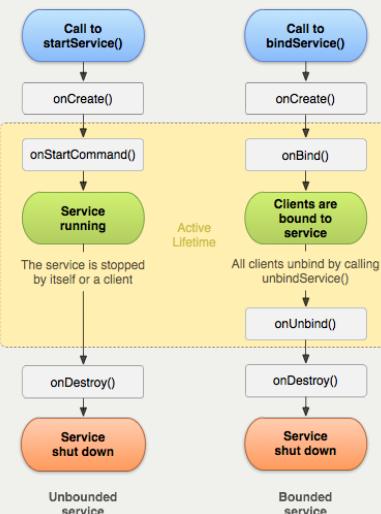
## Utilisation des services

- *Service vs IntentService*
  - **Service**
    - La classe de base de tous les services
    - Les classes implantant directement cette classe doivent créer un thread pour que le code du service n'impacte pas les performances de l'activité qui l'a créé
    - Par défaut, le service s'exécute dans le thread principal
  - **IntentService**
    - Sous-classe de `Service`
    - Services démarrés via un thread qui traite toutes les demandes de démarrage les unes après les autres
    - Intéressant si le service n'a pas besoin de gérer plusieurs requêtes simultanément

## Utilisation des services

- *Service vs Thread*
  - Si une partie du code doit être exécutée en dehors du thread principal et que l'utilisateur doit interagir avec cette partie de code, celle-ci doit être placée dans un thread et pas un service
  - Les classes `AsyncTask` et `HandlerThread` peuvent être utilisées à la place d'un thread

## Cycle de vie d'un service



### • Service « démarré »

- L'invocation de la méthode `startService()` entraîne la création du service et l'invocation de la méthode `onStartCommand()`
- L'invocation de la méthode `stopService()` entraîne celle de la méthode `onDestroy()`

### • Service « lié »

- L'invocation de la méthode `bind()` entraîne celle de la méthode `onBind()`.
- L'invocation de la méthode `unbind()` entraîne celle de la méthode `onUnbind()`

## Un exemple de service

```

public class MyService extends Service {
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //TODO do something useful
        return Service.START_NOT_STICKY;
    }

    @Override
    public IBinder onBind(Intent intent) {
        //TODO for communication return IBinder implementation
        return null;
    }
}

// use this to start and trigger a service
Intent i= new Intent(context, MyService.class);

// potentially add data to the intent
i.putExtra("KEY1", "Value to be used by the service");
context.startService(i);
  
```

## Un exemple de IntentService

```

public class DownloadService extends IntentService {
    public DownloadService() {
        super("DownloadService");
    }

    // called asynchronously by Android
    @Override
    protected void onHandleIntent(Intent intent) {
        String url = intent.getStringExtra(URL)
        // open connection to the URL and download the data
    }
}

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Download" />

    <LinearLayout
        ...
    </LinearLayout>
</LinearLayout>
  
```

## Un exemple de IntentService

```

public class MainActivity extends Activity {
    private TextView textView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView = (TextView) findViewById(R.id.status);
    }

    @Override
    protected void onResume() { super.onResume(); }

    @Override
    protected void onPause() { super.onPause(); }

    public void onClick(View view) {
        Intent intent = new Intent(this, DownloadService.class);
        intent.putExtra(DownloadService.URL, "http://localhost/index.html");
        startService(intent);
        textView.setText("Service started");
    }
}
  
```

## Un exemple de service local

```
public class LocalDataService extends Service {  
    private final IBinder mBinder = new MyBinder();  
  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        return Service.START_NOT_STICKY;  
    }  
  
    @Override  
    public IBinder onBind(Intent intent) {  
        addResultValues();  
        return mBinder;  
    }  
  
    public class MyBinder extends Binder {  
        LocalDataService getService() {  
            return LocalDataService.this;  
        }  
    }  
  
    public List<String> getDataList() {  
        return Arrays.asList("d1", "d2", "d3", "d4");  
    }  
}
```

## Un exemple de service local

```
public class MainActivity extends ListActivity implements ServiceConnection {  
    private LocalDataService s;  
    private ArrayAdapter<String> adapter;  
    private List<String> dataList;  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        dataList = new ArrayList<String>();  
        adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,  
                                         android.R.id.text1, dataList);  
        setListAdapter(adapter);  
    }  
  
    @Override  
    protected void onResume() {  
        super.onResume();  
        Intent intent = new Intent(this, LocalDataService.class);  
        bindService(intent, this, Context.BIND_AUTO_CREATE);  
    }  
}
```

## Un exemple de service local

```
...  
    @Override  
    protected void onPause() {  
        super.onPause();  
        unbindService(this);  
    }  
    @Override  
    public void onServiceConnected(ComponentName name, IBinder binder) {  
        LocalDataService.MyBinder b = (LocalDataService.MyBinder) binder;  
        s = b.getService();  
        Toast.makeText(MainActivity.this, "Connected", Toast.LENGTH_SHORT).show();  
        dataList.clear();  
        dataList.addAll(s.getDataList());  
        adapter.notifyDataSetChanged();  
    }  
  
    @Override  
    public void onServiceDisconnected(ComponentName name) {  
        s = null;  
    }  
}
```

## Intent, IntentFilter et BroadcastReceiver

- Android met en œuvre un système pour diffuser/recevoir des messages dans le système Android et depuis les applications
  - Système similaire au motif de conception *publish/subscribe*
- Les messages sont définis par des instances de la classe *android.content.Intent*
- Un *intent* est constitué
  - d'une action à réaliser
  - d'une donnée à traiter par l'action à réaliser sous forme d'URI (*setData()*) ou d'un type MIME (*setType()*)
  - de paramètres optionnels (définis avec la méthode *putExtra()*)
- *BroadcastReceiver* : composant permettant de recevoir (souscrire) des *intents*
- *IntentFilter* : classe permettant de filtrer les *intents* pour ne s'intéresser qu'à certains d'entre eux

## Modes de diffusion des intents

- 3 types de diffusion des intents :
  - Unicast
    - vers un composant explicitement nommé (par le nom de sa classe Java)
  - Anycast
    - vers un composant assurant une certaine action (visualisation, édition, ...) sur certains types de données
  - Multicast
    - diffusion vers des récepteurs (BroadcastReceiver) qui sont inscrits pour recevoir un type d'intent
- Par défaut, la communication est uni-directionnelle, mais on peut répondre à un intent par un autre intent

## Création des intents

- Intent(String action)
  - Appel implicite pour réaliser une action spécifique
  - Peut être reçu et traité par plusieurs récepteurs potentiels
- Intent(Context, Class<?>)
  - Appel explicite à un composant identifié par sa classe
- Intent(String action, Uri)
  - Appel implicite pour réaliser une action sur la donnée décrite par l'URI

```
// Below an implicit call with an intent defined by a developer  
private static final String MY_BROADCAST = "fr.ubs.BROADCAST_TEST";  
...  
Intent msgToBroadcast = new Intent(MY_BROADCAST);  
sendBroadcast(msgToBroadcast);
```

```
// here an implicit call to an activity with a system-based action and an URI  
Intent telIntent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:36-99"));  
startActivity(telIntent); // Here an explicit call to an activity  
Intent myIntent = new Intent(this, SecondActivity.class);  
myIntent.putExtra("msg", "Mon Message");
```

## Paramétrage des intents

- addCategory(String category)
  - Ajout de catégories apportant des informations supplémentaires sur l'action
- putExtra(String key,value)
  - Ajout de données supplémentaires qui seront utilisées pour réaliser l'action
- setType(String type)
  - Type mime décrivant la donnée
- setFlags(flags)
  - permission sur les données
  - relation activité/BackStack

```
String to = "moi@univ-ubs.fr";  
String subject = "Coucou";  
String msg = "Bonjour à tous";  
  
Intent email = new Intent(Intent.ACTION_SEND);  
email.putExtra(Intent.EXTRA_EMAIL, new String[]{to});  
email.putExtra(Intent.EXTRA_SUBJECT, subject);  
email.putExtra(Intent.EXTRA_TEXT, msg);  
  
email.setType("message/rfc822");  
startActivity(email);
```

## Quelques actions système

- ACTION\_BOOT\_COMPLETED
  - Action déclenchée quand la plateforme mobile a terminé sa séquence de démarrage.
  - Nécessite d'ajouter la permission RECEIVE\_BOOT\_COMPLETED pour recevoir l'intent
- ACTION\_SCREEN\_OFF / ACTION\_SCREEN\_ON
  - Actions diffusée quand l'écran s'allume ou s'éteint
- ACTION\_VIEW content://contacts/people/1
  - affiche des informations concernant le contact ayant comme identifiant 1
- ACTION\_DIAL content://contacts/people/1
  - affiche le dialer avec le numéro de la personne ayant comme identifiant 1

## Quelques actions système (suite)

- ACTION\_DIAL tel:123
  - affiche le dialer avec le numéro prêt à être appelé 123
- ACTION\_EDIT content://contacts/people/1
  - affiche l'édition d'information concernant la personne ayant l'identifiant 1
- De nombreux *intents* sont générés par la plateforme Android pour notifier les applications d'événements comme :
  - La charge/décharge de la batterie
  - La connexion/déconnexion Wi-Fi, 3G/4G, Bluetooth
  - ...
- Ces intents peuvent être captés avec des *BroadcastReceiver*

## BroadcastReceiver

- Pour recevoir les messages diffusés (les *intents*), il y a 2 façons de faire :

- Déclarer des récepteurs dans le fichier *AndroidManifest.xml*
- Crée et enregistrer des récepteurs dans le contexte de l'application

```
public class MyBroadcastReceiver extends BroadcastReceiver {  
    private static final String TAG = "MyBroadcastReceiver";  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        String builder = new StringBuilder();  
        sb.append("Action: " + intent.getAction() + "\n");  
        sb.append("URI: " + intent.toUri(Intent.URI_INTENT_SCHEME).toString() + "\n");  
        Log.d(TAG, sb.toString());  
    }  
    <receiver android:name=".MyBroadcastReceiver" android:exported="true">  
        <intent-filter>  
            <action android:name="ConnectivityManager.CONNECTIVITY_ACTION"/>  
            <action android:name="Intent.ACTION_AIRPLANE_MODE_CHANGED" />  
        </intent-filter>  
    </receiver>
```

```
BroadcastReceiver br = new MyBroadcastReceiver();  
IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);  
filter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);  
this.registerReceiver(br, filter);
```

## BroadcastReceiver

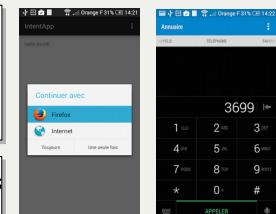
- Si l'enregistrement d'un *BroadcastReceiver* a été fait dans la méthode *onCreate()* d'une activité, alors son désenregistrement doit être fait dans la méthode *onDestroy()* de l'activité
- Si l'enregistrement est fait dans la méthode *onResume()* de l'activité, alors le désenregistrement doit être fait dans la méthode *onPause()*
- Les applications qui ont enregistrées un récepteur de messages dans leur manifest n'ont pas besoin d'être en cours d'exécution pour capter le message
  - Il est possible de lancer ces applications automatiquement et prévoir des comportements en fonction de certains événements qui se produisent sur le mobile sans que l'utilisateur ait à lancer ces dites applications

## Intent et activité

- Un *intent* peut être utilisé pour
  - Démarrer explicitement une activité ou un service en utilisant le nom de la classe
    - Service ou activité locale à l'application
  - Lancer une activité (ou un service) dans le but d'effectuer une action concernant un ensemble de données
  - Lancement implicite d'une activité
    - Demande au système de résoudre l'*intent*. Des applications peuvent répondre à celui-ci

```
Intent telIntent = new Intent(Intent.ACTION_DIAL,  
                                Uri.parse("tel:36-99"));  
startActivity(telIntent);  
  
Intent webIntent = new Intent(Intent.ACTION_VIEW,  
                                Uri.parse("http://www.univ-ubs.fr"));  
startActivity(webIntent);
```

```
Intent intent = new Intent(getApplicationContext(), Activity2.class);  
startActivity(intent);
```



## Lancement implicite d'activité

- Si plusieurs activités ou applications peuvent répondre à l'action demandée, le système proposera alors une liste des applications à l'utilisateur. Sinon, la seule application/activité pouvant traiter la demande sera exécutée
- Pour savoir si une demande pourra être satisfaite, il est possible d'effectuer un test via la méthode `resolveActivity()` qui prend en paramètre un gestionnaire de paquetages (PackageManager).

## IntentFilter

- Pour ne pas recevoir tous les messages diffusés (i.e. tous les intents), il est possible de définir des filtres.
- Les filtres peuvent être définis dans le code ou dans le fichier `AndroidManifest.xml`
- 3 balises sont à prendre en compte dans la balise « `intent-filter` » :
  - `action` : précise le nom de l'action à laquelle répondre
    - Nom de l'intent
    - Pour éviter les collisions, il faut utiliser la convention de nommage de Java
  - `category` : précise dans quelle circonstance une réponse doit être donnée à l'action
    - Il peut y en avoir plusieurs. Elles peuvent être personnalisées ou natives
  - `data`: permet de préciser sur quels types de données le composant peut agir

## Exemples

- Le classique : mon activité déclare une application

```
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

- Mon activité sait lire et éditer les images JPEG

```
<intent-filter android:label="@string/jpeg_editor">
<action android:name="android.intent.action.VIEW" />
<action android:name="android.intent.action.EDIT" />
<data android:mimeType="image/jpeg" />
</intent-filter>
```

- Action personnalisée

```
<activity android:name=".Main" android:label="@string/app_name">
<intent-filter>
<action android:name="fr.ubs.myAction" />
<category android:name="android.intent.category.DEFAULT" />
</intent-filter>
</activity>
```

Button autoinvoc = (Button) findViewById(R.id.autoinvoc);
autoinvoc.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent("fr.ubs.myAction");
startActivity(intent);
}});

## Utilisation des catégories de filtrages

- Les catégories d'intent évoquées précédemment permettent de répondre à des événements dans un contexte particulier.
- Par exemple pour répondre à un clic sur un lien hypertext <ftp://ftp.univ-ubs.fr>, il faut s'y prendre de la manière suivante :

```
<intent-filter>
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="android.intent.category.BROWSABLE" />
<data android:scheme="ftp" android:host="ftp.univ-ubs.fr"/>
</intent-filter>
```

Voir le contenu

Activité qui peut être invoquée depuis un navigateur  
L'activité pourra être déclenchée si on clique sur  
le lien <ftp://ftp.univ-ubs.fr>

## Partage de données via les intents

- Utilisation des « extras » associés aux intents.
- Extras = ensemble de couples clé/valeur
  - La clé est une chaîne de caractères
  - La valeur peut être de type primitif(integer, String, ...) et de type Parcelable.
- Ceci est valable pour les appels implicites et explicites

```
Intent myIntent = new Intent(this, SecondActivity.class);
MyIntent.putExtra("msg", "Mon Message");

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.second_activity);
    Bundle bundle = getIntent().getExtras();
    if (bundle.containsKey("msg")) {
        Log.info("Message:", bundle.getString("msg"));
    } else {
        Log.info("Message: null");
    }
}
```

## Retour d'activité et résultat

- Lorsque le bouton retour est pressé, l'activité courante prend fin et revient à l'activité précédente.
- Au sein d'une application une activité peut vouloir récupérer un code de retour de l'activité « enfant »
- On utilise la méthode `startActivityForResult()` qui envoie un code de retour.
- Il est possible de filtrer le code de retour de l'activité enfant dans la méthode `onActivityResult()`

## Exemple

```
public class MainActivity extends Activity{
    //permet d'identifier l'appelant
    private static final int REQUEST_CODE=12;

    @Override
    protected void onCreate(Bundle savedInstanceState){
        ...
        Intent myIntent = new Intent(this, SecondActivity.class);
        startActivityForResult(myIntent, REQUEST_CODE);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data){
        super.onActivityResult(requestCode, resultCode, data);
        if(requestCode == REQUEST_CODE){
            if(resultCode == RESULT_OK){
                Toast.makeText(this,data.getExtras().getString("msg"), Toast.LENGTH_SHORT).show();
            }else if(resultCode == RESULT_CANCELED){
                Toast.makeText(this,"Activité interrompue", Toast.LENGTH_SHORT).show();
            }
        }
    }
}

public class SecondActivity extends Activity{
    @Override
    protected void onCreate(Bundle savedInstanceState){...}

    @Override
    protected void finish(){
        // On retourne les résultats
        Intent res = new Intent();
        res.putExtra("msg", "Mon message");
        setResult(RESULT_OK,res);
        super.finish();
    }
}
```

## Interfaces graphiques

## Vues et gabarits

- Les éléments graphiques héritent de la classe `android.widget.View`
  - Regroupe l'affichage et la gestion des événements liés à la zone d'affichage
- Les éléments graphiques peuvent être groupés dans un `ViewGroup`
- Des `ViewGroups` particuliers sont prédefinis : les gabarits (*layouts*)
- Quelques gabarits offrant des prédispositions :

Gabarit	Description
LinearLayout	dispose les éléments de gauche à droite ou du haut vers le bas
RelativeLayout	: les éléments enfants sont placés les uns par rapport aux autres
TableLayout	disposition matricielle
GridLayout	disposition matricielle avec N colonnes et un nombre infini de lignes

- Les déclarations se font principalement en XML

## Attributs des gabarits

- Les attributs les plus importants des gabarits sont :
  - `android:layout_width`
  - `android:layout_height`
  - `android:orientation` : définit l'orientation de l'empilement
  - `android:gravity` : définit l'alignement des éléments
- `android:layout_width` et `android:layout_height` peuvent prendre les valeurs :
  - `"match_parent"` : l'élément remplit tout l'élément parent
  - `"wrap_content"` : prend la place minimum nécessaire à l'affichage
  - Une taille fixe définie en pixels par le développeur

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:id="@+id/acceuilid">
    <TextView android:id="@+id/le_texte" android:text="@string/login" />
    <Button android:id="@+id/le_bouton" android:text="@string/bouton" />
</LinearLayout>
```

## Rendu d'une interface par une activité

- Une « interface graphique » définie comme une ressource dans une fichier XML peut être obtenue et rendue par une activité en utilisant les méthodes `findViewById()` et `setContentView()`.

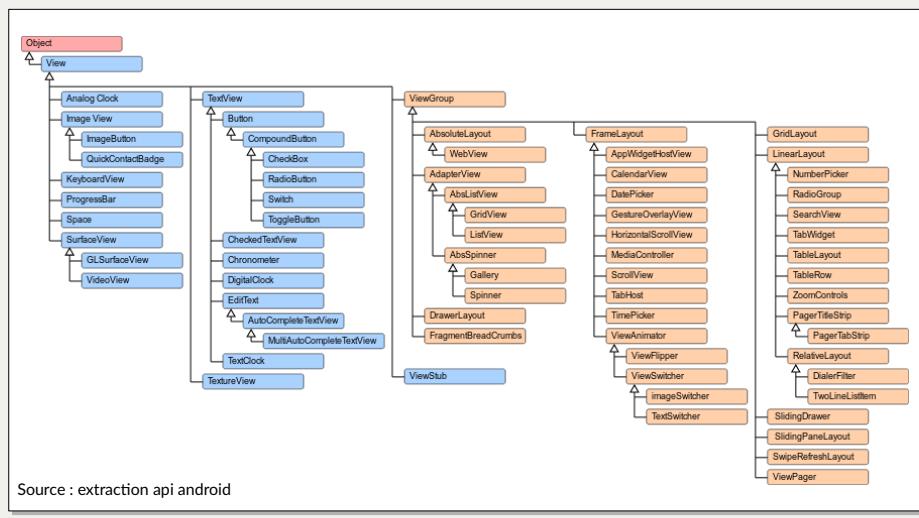
```
public class MyActivity extends Activity{
    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.acceuil);
        TextView tv = findViewById(R.id.le_texte) ;
        tv.setText("hello");
    }
}
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:id="@+id/acceuilid">
    <TextView android:id="@+id/le_texte" android:text="@string/login" />
    <Button android:id="@+id/le_bouton" android:text="@string/bouton" />
</LinearLayout>
```

## Implantation d'une interface avec l'API Java

```
public class Activity2 extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout gabarit = new LinearLayout(this);
        gabarit.setGravity(Gravity.CENTER); // centrer les éléments graphiques
        gabarit.setOrientation(LinearLayout.VERTICAL); // empiler vers le bas !
        TextView texte = new TextView(this);
        texte.setText("Programming creation of interface !");
        gabarit.addView(texte);
        setContentView(gabarit);
    }
}
```

## Les composants graphiques



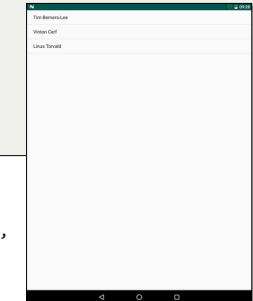
UBS/NLS

Master 2 - INF 4001

72

- Certains composants, comme *ListView*, peuvent afficher des données variées.
    - Les éléments de la liste doivent être insérés dans un *ListAdapter*
    - il faut aussi définir le gabarit qui sera utilisé pour afficher chaque élément du *ListAdapter*

```
<?xml version="1.0" encoding="utf-8"?>
<ListView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```



```
import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
public class MainActivity extends ListActivity {
    private static final String[] data = {"Tim Berners-Lee", "Vinton Cerf",
                                         "Linus Torvald"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, data);
        setListAdapter(adapter);
    }
}
```

UBS/NLS

Master 2 - INF 4001

73

## Liste d'éléments complexes

```
<?xml version="1.0" encoding="utf-8"?>                                activity_main.xml
<ListView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/lst_item">

    <TextView
        android:text="Nom"
        android:id="@+id/nom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </TextView>

    <TextView
        android:text="Prénom"
        android:id="@+id/prenom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </TextView>
</LinearLayout>
```

```
public class Personne {  
    public String nom;  
    public String prenom;  
  
    public Personne(String n, String p){  
        this.nom = n;  
        this.prenom = p;  
    }  
}
```

© UBS/NLS

Master 2 - INF 4001

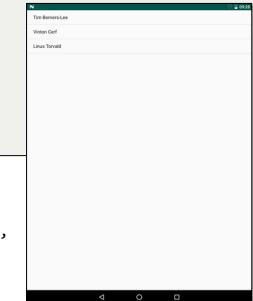
74

# Adaptateurs

- Certains composants, comme *ListView*, peuvent afficher des données variées.

- Les éléments de la liste doivent être insérés dans un *ListAdapter*
- il faut aussi définir le gabarit qui sera utilisé pour afficher chaque élément du *ListAdapter*

```
<?xml version="1.0" encoding="utf-8"?>
<ListView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```



```
import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
public class MainActivity extends ListActivity {
    private static final String[] data = {"Tim Berners-Lee", "Vinton Cerf",
                                         "Linus Torvald"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, data);
        setListAdapter(adapter);
    }
}
```

UBS/NLS

## Liste d'éléments complexes (suite)

```
public class PersonneListAdapter extends BaseAdapter {
    private List<Personne> personneslist;
    private Context context;
    private LayoutInflater inflater;

    public PersonneListAdapter(Context ctx, List<Personne> lst){
        this.personneslist = lst;
        this.context = ctx;
        this.inflater = LayoutInflater.from(ctx);
    }
    @Override
    public int getCount() { return this.personnesList.size(); }

    @Override
    public Object getItem(int position) { return this.personnesList.get(position); }

    @Override
    public long getItemId(int position) { return position; }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View view;
        if(convertView == null){
            //Initialisation de la vue personne list item
            view = (View)inflater.inflate(R.layout.list_item, parent, false);
        }else{
            view = (View) convertView;
        }
        // Initialisation des vues du layout
        TextView nom = (TextView) view.findViewById(R.id.nom);
        TextView prenom = (TextView) view.findViewById(R.id.prenom);
        // modification des vues
        nom.setText(this.personneslist.get(position).nom);
        prenom.setText(this.personnesList.get(position).prenom);
        // On retourne la vue créée
        return view;
    }
}
```

UBS/NLS

Master 2 - INF 4001

75

## Liste d'éléments complexes (suite et fin)

```
MainActivity.java
```

```
public class MainActivity extends Activity {
    private static final ArrayList<Personne> list = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        list.add(new Personne("Berners-Lee", "Tim"));
        list.add(new Personne("Cerf", "Vinton"));
        list.add(new Personne("Torvald", "Linus"));
        //Initialisation de l'adaptateur
        PersonneListAdapter adapter = new PersonneListAdapter(this, list);
        // Récupération de la ListView
        ListView lv = (ListView)findViewById(R.id.list);
        // Affectation des données à la liste
        lv.setAdapter(adapter);
    }

    @Override
    protected void onStart() {
        super.onStart();
    }

    @Override
    protected void onStop() {
        super.onStop();
    }
}
```



## Gestion des interactions utilisateur

- Il existe 2 façons de traiter les événements générés par un élément de type View suite à une action utilisateur :
  - Enregistrement de *listeners* dans le code Java
  - Définition de *listeners* dans le gabarit (*layout*) contenant l'élément de type View
- Plusieurs types de *listeners*
  - View.OnClickListener*, *View.OnKeyListener*, *View.OnDragListener*, *View.OnFocusChangeListener*, *View.OnLongClickListener*, *View.OnTouchListener*, *AdapterView.OnItemClickListener*, *AdapterView.OnItemSelectedListener*, *AdapterView.OnItemLongClickListener*, ...
- Des méthodes spécifiques pour affecter les *listeners* aux éléments de type View
  - setOnClickListener()*, *setOnItemClickListener()*, ...

## Gestion des interactions utilisateur : exemples

```
<?xml version="1.0" encoding="utf-8"?>
...
<Button
    android:id="@+id/signon"
    ...
    android:onClick="signOnClicked" />
...

public class MainActivity extends Activity {
    ...
    public void signOnClicked(View view) {
        Button button = (Button)view;
        Log.e(MAIN_ACTIVITY, button.getId()+" is clicked");
    }
}

public class MainActivity extends Activity{
    ...
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button)findViewById(R.id.signon);
        button.setOnClickListener(new ButtonOnClickListener(button));
    }

    private class ButtonOnClickListener implements View.OnClickListener {
        private final Button button;
        ButtonOnClickListener(Button button) { this.button = button; }
        @Override
        public void onClick(View v) {
            Log.e(MAIN_ACTIVITY, button.getId()+" is clicked");
        }
    }
}

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final Button button = (Button)findViewById(R.id.signon);
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {Log.e(MAIN_ACTIVITY, button.getId()+" is clicked");}
    });
}
```

## Toast

- Les *Toasts* sont des messages courts permettant d'informer l'utilisateur sans perturber le fonctionnement d'une activité (pas d'interruption)
- Les *Toasts* s'effacent après quelques secondes
  - 2 durées : LENGTH\_LONG et LENGTH\_SHORT

```
Toast.makeText(this, "Ceci est un Toast !",Toast.LENGTH_SHORT).show();
```

## Menus

- Il existe 3 types de menus dans une application
  - Menu d'options
    - Menus associés à une activité
    - Ils peuvent être placés dans la barre de l'application (ActionBar ou ToolBar)
  - Menus contextuels
    - Menus « flottant » qui apparaît lorsque l'utilisateur effectue un clic long sur une élément
    - Ils fournissent des actions qui affectent l'élément sélectionné
  - Menu de type popup
    - Menus affichant une liste d'items qui est attaché à la vue qui a invoqué le menu
- Les menus peuvent être définis
  - dans des fichiers XML ressources contenus dans le répertoire /res/menu
  - dans le code Java

## Menus

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new_game"
        android:title="@string/new_game"
        android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```

Precise quand le menu doit apparaître dans la barre de l'application. 4 possibilités :  
- never,  
- always,  
- withText (affiche le texte de l'élément)  
- ifRoom (s'il y a de la place)

```
public class MainActivity extends Activity {
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

## Menus et Toolbar/ActionBar

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MyActionBarActivity">

    <android.support.v7.widget.Toolbar
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:background="#F58FA8"
        android:minHeight="?android:attr/actionBarSize" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_below="@+id/toolbar"
        android:layout_marginTop="10dp"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</RelativeLayout>
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="windowActionBar">false</item>
</style>
```

styles.xml

```
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        setContentView(R.layout.main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }
}
```

MainActivity.java

## WebView

- Vue permettant d'afficher des pages Web (HTML, CSS, JS) au sein d'une application
  - Supporte le zoom comme les navigateurs Web
- Configuration
  - setJavaScriptEnabled() : activation/désactivation du support JavaScript
  - setAllowContentAccess() : autorise/interdit le chargement de contenus depuis les URLs de la page Web chargée
  - setAllowFileAccess() : autoriser/interdit de l'accès aux fichiers locaux
  - ...
- Utilisation
  - loadData(String data, String mimeType, String encoding) : affiche le contenu
  - loadURL(String url) pour charger le contenu de la page désignée par l'URL

## WebView : exemple d'utilisation

```
public class MainActivity extends ActionBarActivity implements OnClickListener {  
    private WebView webView;  
    private EditText url;  
    private Button goBtn;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        this.url = (EditText) this.findViewById(R.id.urlText);  
        this.goBtn = (Button) this.findViewById(R.id.goButton);  
        this.goBtn.setOnClickListener(this);  
  
        WebSettings params = this.webView.getSettings();  
        params.setJavaScriptEnabled(true);  
        params.setBuiltInZoomControls(true);  
        this.webView.addJavascriptInterface(new WebAppInterface(this), "Android");  
        this.webView.setWebViewClient(new MyWebViewClient());  
    }  
  
    @Override  
    public void onClick(View v){  
        if(v.getId() == R.id.goButton){ this.webView.loadURL(this.url.getText().toString());}  
    }  
  
    // This class is used to load content according to URLs specified in the main page that has been  
    // loaded. If such a class is not defined, the ActivityManager will ask user to select a Web  
    // browser to load the content  
    private class MyWebViewClient extends WebViewClient{  
        @Override  
        public boolean shouldOverrideUrlLoading(WebView view, String url){  
            view.loadUrl(url);  
            return true;  
        }  
    }  
}
```

## Programmation concurrente

## Programmation concurrente

- Par défaut, une application s'exécute dans un unique processus
  - Tous les éléments qui composent l'application s'exécutent dans le même processus (*activity*, *service*, *broadcast receiver*, etc.)
- Les traitements ne doivent pas bloquer l'interface graphique pour que l'utilisateur puisse utiliser l'application
  - Certaines opérations ne peuvent pas être réalisées dans « l'UI Thread »
    - ouverture de sockets, établissement de connexions réseau
  - Il faut donc exécuter les traitements de manière concurrente grâce :
    - des processus
    - des threads (Thread, Executor, ThreadPool)
    - des tâches asynchrones (AsyncTask),
    - des tâches périodiques (TimerTask),
    - ...

- Pour qu'un composant (activité, service) puisse s'exécuter dans un autre processus que celui de l'application on utiliser l'attribut *android:process* en préfixant le nom du processus par « `:` »

```
<service android:name=".ConnectionService" android:process=":p2">
```

- Le système crée un thread d'exécution pour l'application : le **thread principal**
  - Le thread principal est notamment responsable de l'interface graphique
- Pour éviter les blocages, il est possible de déléguer à « l'UI Thread » l'exécution ultérieure d'un traitement

## Thread et interface graphiques : exemples

```
public void onClick(View v) { // DO NOT DO THIS !
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }).start(); }
```

Ce code peut entraîner une exception  
CalledFromWrongThreadException

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap =
                loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }).start(); }
```

La méthode View.post(Runnable) permet  
d'éviter cette exception  
Les méthodes suivantes peuvent aussi être utilisées  
•Activity.runOnUiThread(Runnable)  
•View.postDelayed(Runnable, long)

## AsyncTask

- AsyncTask permet d'exécuter des opérations en arrière plan
- Les résultats sont retournés dans le thread principal en fin d'exécution de la tâche
- AsyncTask est adaptée pour des traitements courts, dont la progression et les résultats doivent être ensuite affichés sur l'interface utilisateur
- Attention, ces tâches ne persistent pas entre les redémarrages de l'activité.
  - Si l'orientation de la plateforme mobile change, la tâche asynchrone sera annulée.

## AsyncTask : exemple

```
// tâche téléchargeant une liste de fichiers identifiés par des URLs
// retourne la taille des données téléchargées quand la tâche se termine
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

## Timer, TimerTask et interface utilisateur

- Les classes Timer et TimerTask permettent de définir et d'exécuter périodiquement des tâches

```
Timer timer = new Timer();
TimerTask task = new TimerTask() {
    public void run() { Log.i(TAG, "Time:"+System.currentTimeMillis()); }
};
timer.schedule(task, 1000);
```

- Attention, le code précédent ne fonctionne que si on n'interagit pas avec l'interface graphique. Sinon, il faut utiliser la méthode runOnUiThread() ou une classe Handler par exemple.

```
final Handler handler = new Handler();
TimerTask task = new TimerTask() {
    public void run() {
        handler.post(new Runnable() {
            public void run() { Toast.makeText(MainActivity.this, "hello !", Toast.LENGTH_SHORT).show(); }
        });
    }
};
timer.schedule(task, 0, 5000);

Timer task = new TimerTask() {
    public void run() {
        MainActivity.this.runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(MainActivity.this, "hello !", Toast.LENGTH_SHORT).show();
            }
        });
    }
};
timer.schedule(task, 0, 5000);
```

## Développement d'applications mobiles hybrides - INF4001 -

Nicolas Le Sommer

Nicolas.Le-Sommer@univ-ubs.fr

Université de Bretagne Sud, IUT de Vannes, Département Informatique

### Plan du cours

1. Applications mobiles hybrides Web vs Natives
2. Développement d'applications mobiles hybrides Web
  1. Apache Cordova
  2. Ionic
3. Développement d'applications mobiles hybrides natives
  1. ReactNative
  2. NativeScript

Applications mobiles hybrides multi-plateformes



# Présentation de Cordova

- Cordova est un framework de développement d'applications mobiles libre développé dans le projet Apache
  - <https://cordova.apache.org/>
- Il permet de développer une application multi-platformes en utilisant des technologies Web (HTML5, CSS3, Javascript)
  - N'importe quel framework de développement Web peut être utilisé
    - Angular, AngularJS, ReactJS, VueJS, ...
- L'application produite est une application native qui « enrobe » une application Web qui s'exécute dans un composant natif de type vue Web
  - WebView ou AndroidSystemWebView pour Android
  - Pas de zoom, pas de barre d'URL



# Présentation de Cordova (suite et fin)

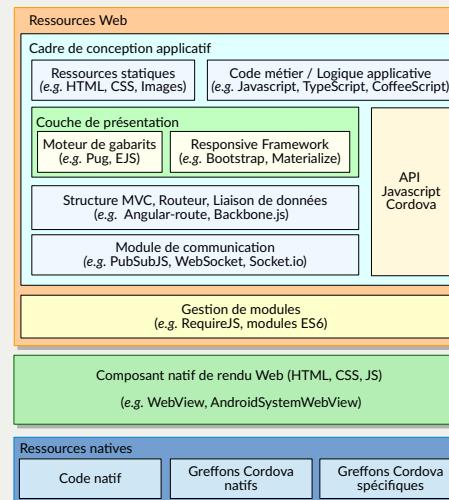
- Des greffons permettent de faire le lien entre le code Javascript et les APIs natives des différents systèmes
  - Localisation, appareil photo, batterie, contacts, ...
  - Possibilité de développer ses propres greffons
- Cordova permet de produire des applications pour différentes cibles

	Amazon-fireos	Android	blackberry10	Firefox OS	iOS	Ubuntu	wp8 (Windows Phone 8)	Windows (8.0, 8.1, 10, Téléphone 8.1)	paciarelli
Cordova CLI	✓ Mac, Windows, Linux	✓ Mac, Windows, Linux	✓ Mac, Windows	✓ Mac, Windows, Linux	✓ Mac	✓ Ubuntu	✓ Windows	✓	✗
Embedded WebView	✓ (voir détails)	✓ (voir détails)	✗	✗	✓ (voir détails)	✓	✗	✗	✗
Plugin Interface	✓ (voir détails)	✓ (voir détails)	✓ (voir détails)	✗	✓ (voir détails)	✓	✓ (voir détails)	✓	✗

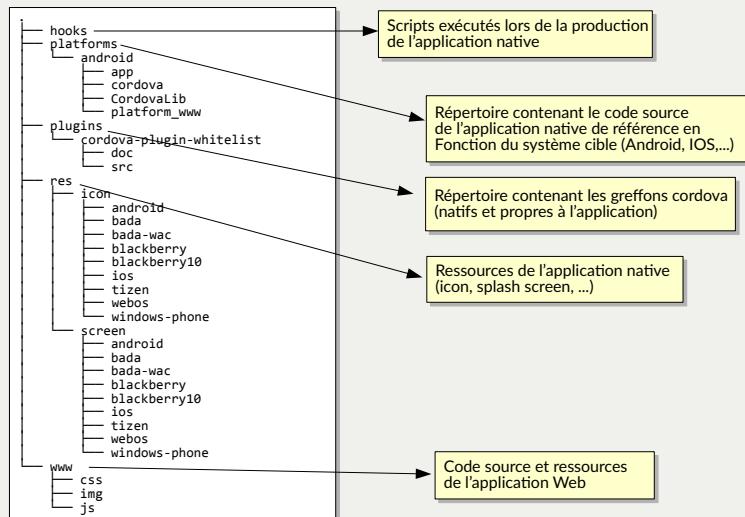
## Support des greffons

	Amazon-fireos	Android	blackberry10	Firefox OS	iOS	Ubuntu	wp8 (Windows Phone 8)	Windows (8.0, 8.1, 10, Téléphone 8.1)	paciarelli
Accéléromètre	✓	✓	✓	✓	✓	✓	✓	✓	✓
BatteryStatus	✓	✓	✓	✓	✓	✗	✓	✓	✓
Appareil photo	✓	✓	✓	✓	✓	✓	✓	✓	✓
Capture	✓	✓	✓	✗	✓	✓	✓	✓	✗
Boussole	✓	✓	✓	✗	✓	✓	✓	✓	✓
Connexion	✓	✓	✓	✗	✓	✓	✓	✓	✓
Contacts	✓	✓	✓	✓	✓	✓	✓	partiellement	✗
Dispositif	✓	✓	✓	✓	✓	✓	✓	✓	✓
Événements	✓	✓	✓	✗	✓	✓	✓	✓	✓
Fichier	✓	✓	✓	✗	✓	✓	✓	✓	✗
Transfert de fichiers	✓	✓	✗	✗	✓	✗	✗	✗	✗
Géolocalisation	✓	✓	✓	✓	✓	✓	✓	✓	✓
Mondialisation	✓	✓	✓	✗	✓	✓	✓	✓	✗
InAppBrowser	✓	✓	✓	✗	✓	✓	✓	utilise les iframes	✗
Media	✓	✓	✓	✗	✓	✓	✓	✓	✓
Notification	✓	✓	✓	✗	✓	✓	✓	✓	✓
SplashScreen	✓	✓	✓	✗	✓	✓	✓	✓	✗
Barre d'État	✗	✓	✗	✗	✓	✗	✓	✓	✗
Stockage	✓	✓	✓	✗	✓	✓	✓	localStorage & indexedDB	✓
Vibration	✓	✓	✓	✓	✓	✗	✓	✓	✗

## Architecture générale



## Structure d'une application Cordova



## Le fichier config.xml

- Le fichier config.xml est le fichier de configuration d'une application Cordova
- Attention, ce fichier peut être modifié par la commande en ligne cordova lors de l'ajout d'une plateforme, d'un plugin, ...
- Le fichier config.xml
  - définit l'identifiant de l'application, sa version et son nom
  - fournit une description de l'application
  - liste les plateformes cibles
  - définit les préférences pour les applications natives en fonction des systèmes cibles
  - indique les icons et les splash screen à utiliser
  - précise les permissions éventuelles en fonction du système visé
  - répertorie les greffons à utiliser et à installer
  - liste les scripts (hooks) à exécuter lors de la production des applications natives

## Le fichier config.xml : exemple

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="fr.ubs.myApp" version="1.0.0"
      xmlns="http://www.w3.org/ns/widgets"
      xmlns:cdv="http://cordova.apache.org/ns/1.0">
  <name>myApp</name>
  <description>
    Hello world application!
  </description>
  <author email="nicolas.le-sommer@univ-ubs.fr">
    NLS
  </author>
  <content src="index.html" />
  <plugin name="cordova-plugin-whitelist" spec="1" />
  <access origin="*.*" />
  <allow-intent href="http:///*/*" />
  <allow-intent href="https:///*/*" />
  <allow-intent href="tel:/*" />
  <allow-intent href="sms:/*" />
  <allow-intent href="mailto:/*" />
  <allow-intent href="geo:/*" />
  <platform name="android">
    <allow-intent href="market:/*" />
    <preference name="KeepRunning" value="false" />
    <preference name="LoadUrlTimeoutValue" value="10000" />
    <preference name="InAppBrowserStorageEnabled" value="true" />
    <preference name="ShowTitle" value="true" />
    <preference name="LogLevel" value="VERBOSE" />
    <preference name="AndroidLaunchMode" value="singleTop" />
    <preference name="AndroidPersistentFileLocation" value="Compatibility" />
    <preference name="ErrorUrl" value="www.cordova.error.html" />
    <icon density="ldpi" src="res/android/ldpi.png" />
    <icon density="mdpi" src="res/android/mdpi.png" />
    <icon density="hdpi" src="res/android/hdpi.png" />
    <icon density="xhdpi" src="res/android/xhdpi.png" />
    <icon density="xxhdpi" src="res/android/xxhdpi.png" />
    <icon density="xxxhdpi" src="res/android/xxxhdpi.png" />
  </platform>
</widget>
```

## Le fichier config.xml : exemple (suite et fin)

```
<splash density="land-hdpi" src="res/android/splash-land-hdpi.png" />
<splash density="land-ldpi" src="res/android/splash-land-ldpi.png" />
<splash density="land-mdpi" src="res/android/splash-land-mdpi.png" />
<splash density="land-xhdpi" src="res/android/splash-land-xhdpi.png" />
<splash density="port-hdpi" src="res/android/splash-port-hdpi.png" />
<splash density="port-ldpi" src="res/android/splash-port-ldpi.png" />
<splash density="port-mdpi" src="res/android/splash-port-mdpi.png" />
<splash density="port-xhdpi" src="res/android/splash-port-xhdpi.png" />
<preference name="AutoHideSplashScreen" value="false" />
<preference name="SplashMaintainAspectRatio" value="true" />
<preference name="SplashShowOnlyFirstTime" value="true" />
<preference name="LoadingDialog" value="MyApp, loading..." />
<preference name="android-minSdkVersion" value="21" />
<preference name="android-targetSdkVersion" value="21" />
<hook src="hooks/1_change_java_version.sh" type="before_compile" />
<hook src="hooks/2_Cordova-icon-splash.sh" type="before_platform_add" />
</platform>
<platform name="ios">
  <allow-intent href="itms:/*" />
  <allow-intent href="itms-apps:/*" />
</platform>
<preference name="fullscreen" value="false" />
<preference name="DisallowOverscroll" value="false" />
<preference name="Orientation" value="default" />
<preference name="HideKeyboardFormAccessoryBar" value="false" />
<preference name="BackgroundColor" value="0xffffffff" />
<plugin name="cordova-plugin-whitelist" spec="1.3.3" />
<plugin name="cordova-plugin-camera" spec="3.0.0" />
<plugin name="cordova-plugin-file" spec="5.0.0" />
<plugin name="cordova-plugin-splashscreen" spec="4.1.0" />
<plugin name="myapp-plugin" spec="myapp-plugin" />
<engine name="android" spec="~7.1.1" />
</widget>
```

## Exemple de fichier index.html

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>C3PO Application</title>
    <!-- css -->
    <link rel="stylesheet" type="text/css" href="css/style.css" />

    <!-- angularjs -->
    <script type="text/javascript" src="lib/angular/angular.min.js"></script>
    <script type="text/javascript" src="lib/angular-route/angular-route.min.js"></script>

    <!-- cordova script -->
    <script type="text/javascript" src="cordova.js"></script>
    <script type="text/javascript" src="js/app.js"></script>
    <script type="text/javascript" src="js/event.js"></script>
  </head>
  <body ng-app="app">

    <div class="container bs-docs-container">
      <div role="main">
        <h2>list of events</h2>
        <ul>
          <li ng-repeat="evt in events">
            {{evt}}
          </li>
        </ul>
      </div>
    </div>
  </body>
</html>
```

app.controller('EventController', function (\$scope){
 var eventman = cordova.require('app-plugin.EventMan');

 post = function(event) {
 eventman.postEvent(event, nullCallbackHandler, errorMsgPopup);
 };

 register = function(topics, handler){
 eventman.addHandler(topics, handler, errorMsgPopup)
 };

 register(['event'], new function(evt){
 \$scope.events.push(JSON.stringify(event));
 });
});

## Les hooks

- Les *hooks* sont des scripts qui peuvent être ajoutés par l'application ou les greffons pour « spécialiser » les commandes de gestion du cycle de vie de production de l'application
- Exemples de *hooks*

Type de hook	Commande Cordova associée	Description
before_platform_add	cordova platform add	Script exécuté avant et après l'ajout d'une plateforme
after_platform_add		
before_prepare	cordova prepare	
after_prepare	cordova platform add	Script exécuté avant et après la préparation de l'application
before_compile	cordova compile	
after_compile	cordova build	Script exécuté avant et après la compilation de l'application
before_build	cordova build	
after_build		Script exécuté avant et après la construction de l'application
before_plugin_add	cordova plugin add	
after_plugin_add		Script exécuté avant et après l'ajout d'un greffon

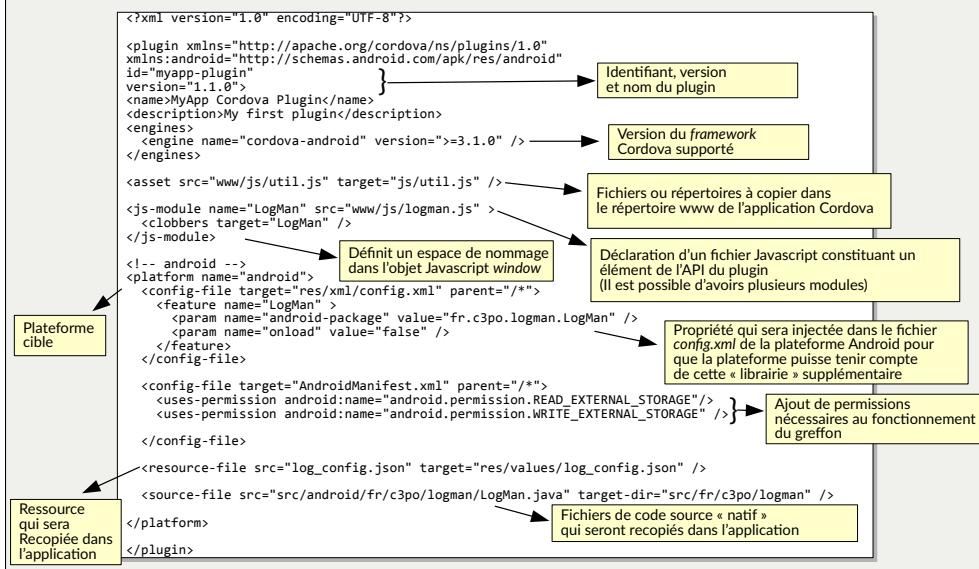
## Cordova-CLI

- L'interface de ligne de commande pour Cordova (Cordova-CLI) est distribuée via un paquetage *npm* (paquetage Node.js)
- La commande *cordova* permet
  - de créer une application (à partir d'un modèle)
  - d'ajouter/retirer des plateformes
    - c'est-à-dire, d'ajouter/supprimer les sources pour construire des applications natives pour les plateformes cibles (**Browser**, **Android**, **IOS**, ...)
  - d'ajouter/retirer des greffons
  - de compiler les sources de l'application
  - de déployer l'application sur un terminal
  - d'exécuter l'application dans un serveur local

## Greffons : présentation

- Un greffon permet de faire le lien entre le code métier de l'application développé en Javascript et l'API native du système cible
- Un greffon
  - est décrit par un fichier *plugin.xml*
  - fournit une interface de programmation en Javascript
    - quelque soit le système cible
  - contient du code natif exploitant l'API des systèmes cibles
    - Dans un greffon, il peut y avoir du code pour **Android**, **IOS**, **FirefoxOS**, ...
- Il existe des greffons « officiels » Cordova
  - Location, Notification, Camera, ...
- Il existe aussi un dépôt de greffons
  - <https://cordova.apache.org/plugins/>

## Greffons : exemple de fichier plugin.xml



## Greffons : exemple d'interface Javascript

- La fonction exec() Javascript de Cordova à 5 paramètres :
  - 2 callbacks (succès et erreur), le service, l'action et des paramètres

```
var exec = require('cordova/exec');

function LogMan() {}

LogMan.prototype.verbose = function (tag, msg, successCallback, errorCallback) {
  exec(successCallback, errorCallback, 'LogMan', 'VERBOSE', [tag, JSON.stringify(msg)]);
};

LogMan.prototype.debug = function (tag, msg, successCallback, errorCallback) {
  exec(successCallback, errorCallback, 'LogMan', 'DEBUG', [tag, JSON.stringify(msg)]);
};

LogMan.prototype.info = function (tag, msg, successCallback, errorCallback) {
  exec(successCallback, errorCallback, 'LogMan', 'INFO', [tag, JSON.stringify(msg)]);
};

LogMan.prototype.warning = function (tag, msg, successCallback, errorCallback) {
  exec(successCallback, errorCallback, 'LogMan', 'WARNING', [tag, JSON.stringify(msg)]);
};

LogMan.prototype.error = function (tag, msg, successCallback, errorCallback) {
  exec(successCallback, errorCallback, 'LogMan', 'ERROR', [tag, JSON.stringify(msg)]);
};

var logman = new LogMan();
module.exports = logman;
```

## Greffon : exemple de code natif Android

- Les classes Java définissant les greffons pour Android étendent la classe *CordovaPlugin*

```
public final class LogMan extends CordovaPlugin {
    private enum Method {VERBOSE, DEBUG, INFO, WARNING, ERROR}

    public LogMan() {}

    @Override
    public void initialize(CordovaInterface cordova, CordovaWebView webView) {
        super.initialize(cordova, webView);
    }

    @Override
    public void asyncExec(String action, JSONArray args, CallbackContext callbackContext)
        throws JSONException {
        Method method = Method.valueOf(action);
        String tag = args.getString(0);
        String msg = args.getString(1);
        switch (method) {
            case VERBOSE: Log.verbose(tag, msg); break;
            case DEBUG: Log.debug(tag, msg); break;
            case INFO: Log.info(tag, msg); break;
            case WARNING: Log.warning(tag, msg); break;
            case ERROR: Log.error(tag, msg); break;
        }
        PluginResult result = new PluginResult(PluginResult.Status.OK);
        result.setKeepCallback(true); // to allow multiple callbacks
        callbackContext.sendPluginResult(result);
    }
}
```

## Greffons : partie native (suite)

- Un greffon ne s'exécute pas dans le thread principal de la WebView
  - Il s'exécute dans un thread exécutant le code principal du framework cordova (WebCore)
- Si le greffon doit accéder à l'interface utilisateur, on doit utiliser la méthode *runOnUiThread()* d'Android

```
@Override
public boolean execute(String action, JSONArray args, final CallbackContext callbackContext)
    throws JSONException {
    Method method = Method.valueOf(action);
    String tag = args.getString(0);
    String msg = args.getString(1);
    switch (method) {
        case VERBOSE: Log.verbose(tag, msg); break;
        case DEBUG: Log.debug(tag, msg); break;
        case INFO: Log.info(tag, msg); break;
        case WARNING: Log.warning(tag, msg); break;
        case ERROR: Log.error(tag, msg); break;
    }
    cordova.getActivity().runOnUiThread(new Runnable() {
        public void run() {
            ...
            callbackContext.success(); // Thread-safe.
        }
    });
    ...
}
```

## Greffons : partie native (suite et fin)

- Pour ne pas bloquer le « WebCore » on peut créer un autre thread via le pool de thread de Cordova.

```
@Override  
public boolean execute(String action, JSONArray args, final CallbackContext callbackContext)  
throws JSONException {  
    cordova.getThreadPool().execute(new Runnable() {  
        public void run() {  
            callbackContext.success(); // Thread-safe.  
        }  
    });  
    ...  
}
```

- Cycle de vie des greffons :

- Une instance d'un plugin est créée pour toute la vie de chaque WebView
- Un plugin n'est instancié qu'au moment de leur premier référencement et appel à une méthode Javascript
  - Sauf si le paramètre *onload* vaut *true* dans le fichier config.xml/plugin.xml

## Stockage : WebSQL

- WebSQL offre une API pour stocker des données dans une base de données structurée
  - Standardisée par le W3C
- La base de données peut être interrogée via des requêtes SQL « standards »
  - Base de données SQLite généralement
- Supportée par les plateformes IOS et Android
- Avantages :
  - Bonnes performances (données peuvent indexées pour des recherches rapides, API asynchrone et non bloquant), modèle transactionnel
- Inconvénients :
  - Pas supporté par toutes les plateformes, API dépréciée, quantité de stockage limitée (~5Mo), structures de données rigides (tables)

## Stockage des données : LocalStorage

- 4 possibilités pour stocker des données produites par l'application Web : LocalStorage, WebSQL, IndexDB et FileSystem

- LocalStorage

- Stockage permettant de stocker des couples clé/valeur

- Avantages :

- Supporté par les WebView utilisées par Cordova et par toutes les plateformes
- Simple d'utilisation
- API synchrone

```
var storage = window.localStorage;  
storage.setItem("key1", "value1");  
var value = storage.getItem("key1");  
storage.removeItem("key1");
```

- Inconvénients :

- Stockage de chaînes de caractères (les structures de données complexes doivent être sérialisées)
- Performances médiocres sur des quantités de données importantes
  - sérialisation/déserialisation, pas d'indexation
- Peu de capacités de stockage (~5Mo)

## Stockage : IndexedDB

- IndexedDB reprend les points forts de LocalStorage et de WebSQL, mais sans leurs points faibles
- Avantages :
  - Des objets Javascript arbitraires peuvent être stockés et référencés par une clé
  - Pas de structure à définir préalablement
  - Possibilité de créer plusieurs bases de données
  - API asynchrone et non bloquante
- Inconvénients :
  - API complexe avec des callbacks imbriqués
  - Supportée par les plateformes Cordova Android et Windows et pas IOS
  - Capacité de stockage limitée (~5Mo)

## Stockage : FileSystem API

- FileSystem API est une spécification du W3C implantée uniquement par le navigateur Web Chrome
- Pour toutes les autres plateformes, il faut utiliser le greffon File de Cordova
- Greffon permettant de lire et écrire des données sur le système de fichiers
- Différentes zones de stockage
  - cordova.file.applicationDirectory,
  - cordova.file.applicationStorageDirectory,
  - cordova.file.dataDirectory,
  - cordova.file.cacheDirectory,
  - cordova.file.externalDataDirectory,
  - ...

## Cordova: références

- <https://cordova.apache.org/>
- Greffons
  - [https://cordova.apache.org/docs/en/8.x/plugin\\_ref/spec.html](https://cordova.apache.org/docs/en/8.x/plugin_ref/spec.html)
  - <https://cordova.apache.org/docs/en/8.x/guide/hybrid/plugins/index.html>
  - <https://cordova.apache.org/docs/en/8.x/guide/platforms/android/plugin.html>
- Stockage
  - <https://cordova.apache.org/docs/en/8.x/cordova/storage/storage.html>



## Présentation de Ionic

- Ionic est un cadre de conception pour produire des applications hybrides multi-plateformes
- Ionic repose sur Cordova ou Capacitor
- Les premières versions d'Ionic étaient fondées sur AngularJS et Javascript
- Les versions récentes utilisent TypeScript et permettent un développement en utilisant Angular, Vue.js ou React.js
- Ionic offre un ensemble de composants graphiques pour faciliter le développement d'applications mobiles
  - Boutons, cartes, checkbox, grille, menus, barre d'outils, ...
- Il propose également une API TypeScript pour
  - manipuler les composants graphiques (propriétés, événements, ...)
  - gérer l'application
    - MenuController, Platform, Config, LoadingController,...

## Présentation de Ionic (suite et fin)

- Ionic Native enrobe en TypeScript les greffons Cordova ou Capacitor qui fournissent un accès au système natif
- Ionic permet de personnaliser facilement les applications en définissant des styles CSS via Sass
  - Sass est un langage de style CSS définissant des variables, des boucles et *mixins* (parties de style pouvant être réutilisées)
- Ionic fournit des icons pour développer les applications
  - Ils sont adaptés au système cible
- Ionic-CLI
  - L'interface de ligne de commande de Ionic étend celle de Cordova
    - On retrouve des commandes similaires

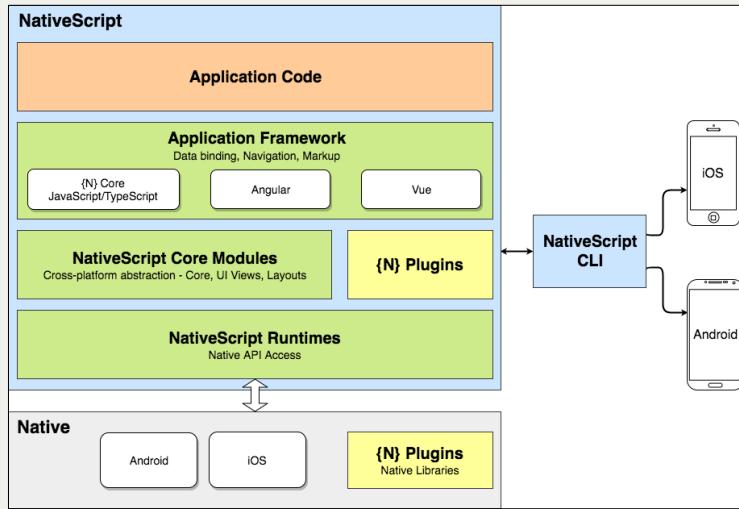
## Applications mobiles natives multi-plateformes



## Présentation de NativeScript

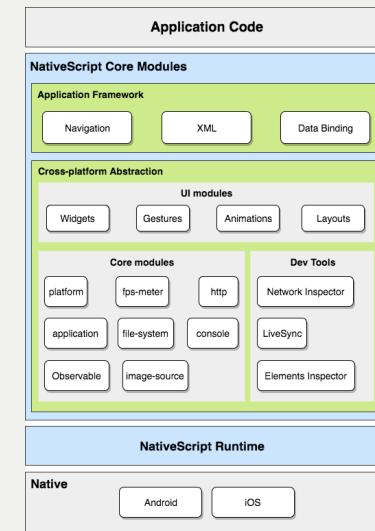
- NativeScript est un cadre de conception et un ensemble d'outils pour développer des applications mobiles multi-plateformes
- Il permet de définir
  - en XML les interfaces graphiques de l'application
  - en Javascript/TypeScript le code métier, accéder aux données et définir le flux d'exécution de l'application
  - en CSS pour définir le style de l'application
- Le code Javascript est interprété à la volée par les machines virtuelles
  - Chrome V8 sur Android
  - Webkit/JavascriptCore sur IOS
- NativeScript permet d'invoquer les API natives Android et IOS à partir de Javascript
- NativeScript peut être combiné avec Angular et Vue.js, mais il peut aussi être utilisé seul

## Vue d'ensemble de NativeScript et de son fonctionnement



Source : <https://docs.nativescript.org>

## Modules composant NativeScript



Source : <https://docs.nativescript.org>

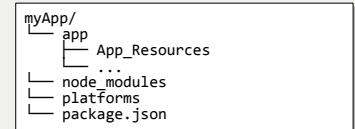
## NativeScript-CLI

- NativeScript et son interface de ligne de commande (NativeScript-CLI) s'installent via le gestionnaire de paquetage de Node.js (*npm*)
- L'interface de ligne de commande permet
  - de créer un nouveau projet
  - d'ajouter/de retirer une plateforme cible
  - de construire l'application pour une cible donnée
  - de déployer et d'exécuter l'application sur un terminal (ou un émulateur)
- L'interface de ligne de commande de NativeScript se nomme *tns*

```
npm install -g nativescript
tns create myApp
tns platform add android
tns build android
tns run
```

## Structure d'une application

- Hiérarchie de répertoires
  - Le répertoire *app* contient les sources de l'application
  - Le répertoire *platforms* contient les sources des applications natives et les sources compilées, les APKs,
- Une application NativeScript est composée d'un ensemble de pages représentant les différents écrans de l'application
- Les pages sont des fichiers XML qui intègrent des composants graphiques (*Layouts*, *Widgets*)
- Le module *Frame* définit des méthodes permettant de naviguer entre les pages



## Création d'une page

- Création d'une page en XML

```
<Page loaded="onPageLoaded">
  <Page.actionBar>
    <ActionBar title="Page Creation"/>
  </Page.actionBar>
  <!-- Each page can have only a single root view -->
  <StackLayout>
    <!-- Content here -->
    <Label text="Hello, world!"/>
  </StackLayout>
</Page>
```

```
function onPageLoaded(args) {
  console.log("Page Loaded");
  const page = args.object;
}
exports.onPageLoaded = onPageLoaded;
```

- Création d'une page via le code Javascript

```
const Page = require("tns-core-modules/ui/page").Page;
const Label = require("tns-core-modules/ui/label").Label;
const StackLayout = require("tns-core-modules/ui/layouts/stack-layout").StackLayout;
function createPage() {
  const stack = new StackLayout();
  const label = new Label();
  label.text = "Hello, world!";
  stack.addChild(label);
  const page = new Page();
  // A page can have only one single root view set
  // via the content property (in this case a StackLayout)
  page.content = stack;
  return page;
}
exports.createPage = createPage;
```

## Définition de la page principale et navigation entre les pages

- Définition du point d'entrée (page principale) de l'application

```
// app.js
const application = require("tns-core-modules/application");
application.run({ moduleName: "app-root" });
```

```
<!-- app-root.xml -->
<Frame defaultPage="home/home-page" />
```

```
<!-- home/home-page.xml -->
<Page class="page">
  <StackLayout>
    <!-- content here-->
    <Label text="Hooray! Home Page loaded!"/>
  </StackLayout>
</Page>
```

- Exemple avec un composant TabView

```
<!-- app-root.xml -->
<TabView androidTabsPosition="bottom">
  <TabViewItem title="First">
    <Frame defaultPage="home/home-page" />
  </TabViewItem>
  <TabViewItem title="Second">
    <Frame defaultPage="second/second-page" />
  </TabViewItem>
</TabView>
```

## Navigation entre les pages (suite)

- Pour obtenir une référence sur l'instance de Frame en cours d'utilisation, on peut utiliser
  - la méthode `topmost()` du module `Frame`
  - la méthode `getFrameById()` du module `Frame` et indiquer le nom de la `frame` recherchée
  - la propriété `frame` d'une instance de `Page`
- Pour naviguer entre les pages, on utilise la fonction `navigate()` de `Frame`
  - Prend le nom de la page
  - Une fonction retournant une page
  - Une page avec un contexte

```
const frameModule = require("tns-core-modules/ui/frame");
const topmostFrame = frameModule.topmost();

const firstFrame = frameModule.getFrameById("firstFrame");
const Page = require("tns-core-modules/ui/page").Page;
page.frame.navigate("second/second-page");
```

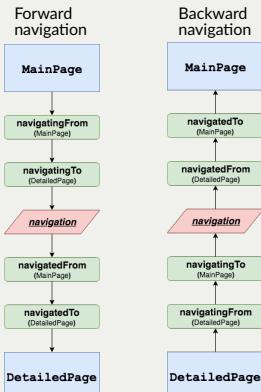
## Navigation entre les pages (suite et fin)

- Les pages visitées par l'utilisateur sont placées dans une pile
- Pour revenir à la page précédente, il faut utiliser la méthode `goBack()` de `Frame`

```
const topmost = require("tns-core-modules/ui/frame").topmost;
topmost.goBack();
```

## Événements associés aux pages

```
<Page navigatedFrom="onNavigatedFrom"
navigatedTo="onNavigatedTo"
loaded="onPageLoaded"
navigatingFrom="onNavigatingFrom"
navigatingTo="onNavigatingTo"
unloaded="onUnloaded"
layoutChanged="onLayoutChanged">
</Page>
```



Source : <https://docs.nativescript.org>

```
function onNavigatingTo(args) {
    console.log(args.eventName);
    console.log(args.object);
    console.log(args.context);
    console.log(args.isBackNavigation);
}
exports.onNavigatingTo = onNavigatingTo;

function onPageLoaded(args) {
    console.log(args.eventName);
    console.log(args.object);
}
exports.onPageLoaded = onPageLoaded;

function onLayoutChanged(args) {
    console.log(args.eventName);
    console.log(args.object);
}
exports.onLayoutChanged = onLayoutChanged;

function onNavigatedTo(args) {
    console.log(args.eventName);
    console.log(args.object);
    console.log(args.context);
    console.log(args.isBackNavigation);
}
exports.onNavigatedTo = onNavigatedTo;

...
```

## Les modules NativeScript

- Core Modules (liste non exhaustive)

Module	Description
application	Abstraction de l'application avec des fonctions de manipulation associées
console	Module permettant d'afficher des traces dans la console utilisateur
http	Module permettant d'envoyer et recevoir des requêtes/réponses HTTP
connectivity	Module fournissant des informations sur la connectivité réseau

- Functionality Modules (liste non exhaustive)

Module	Description
platform	Module fournit des informations sur le terminal, le système et les logiciels
file-system	Module permettant d'accéder au système de fichiers, de lire et écrire des fichiers

- Interface utilisateur (liste non exhaustive)

Module	Description
frame	Fournit la classe Frame qui représente une vue logique et qui permet la navigation dans l'application
page	Fournit la classe Page qui représente une unité logique de navigation dans une frame

## Les modules NativeScript (suite)

- Liaison de données

Module	Description
data/observable	Classe Observable représentant un objet observable ou une données dans le paradigme MVVM
data/observable-array	Classe ObservableArray réagissant aux changements intervenant dans une collection d'objets
data/virtual-array	Classe VirtualArray fonctionnant comme ObservableArray mais permettant de charger des données à la demande

- Les layouts

- Stack-layout, grid-layout, absolute-layout, wrap-layout et dock-layout

- Les widgets

- Button, label, text-field, text-view, list-view, image, scroll-view, date-picker, time-picker, web-view, slider, progress, ...

## Cycle de vie d'une application

- NativeScript fournit le moyen de gérer le cycle de vie de l'application à travers le module *application*

- Démarrage d'application via la méthode *run()* (obligatoire)

```
const application = require("tns-core-modules/application");
application.run({ moduleName: "app-root" });
```

- Événements de l'application

Événement	Description
launch	Événement généré lors du lancement de l'application
suspend	Événement généré lorsque l'application est suspendue
resume	Événement généré lorsque l'application est réactivée après avoir été suspendue
displayed	Événement généré lorsque les éléments d'interface utilisateur sont affichés
exit	Événement généré lorsque l'application est sur le point de s'arrêter
lowMemory	Événement généré lorsqu'il reste peu de mémoire sur le terminal mobile
uncaughtError	Événement généré lorsqu'une erreur non captée est générée

## Cycle de vie d'une application : exemple

```
const application = require("tns-core-modules/application");
application.on(application.launchEvent, (args) => {
    if (args.android) {
        // For Android applications, args.android is an android.content.Intent class.
        console.log("Launched Android application with the following intent: " + args.android + ".");
    }
});

application.on(application.suspendEvent, (args) => {
    if (args.android) {
        // For Android applications, args.android is an android activity class.
        console.log("Activity: " + args.android);
    }
});

application.on(application.displayedEvent, (args) => {
    // args is of type ApplicationEventData
    console.log("displayedEvent");
});

application.on(application.orientationChangedEvent, (args) => {
    // args is of type OrientationChangedEventArgs
    console.log(args.newValue); // "portrait", "landscape", "unknown"
});

application.on(application.exitEvent, (args) => {
    if (args.android) {
        // For Android applications, args.android is an android activity class.
        console.log("Activity: " + args.android);
    } else if (args.ios) {
        // For iOS applications, args.ios is UIApplication.
        console.log("UIApplication: " + args.ios);
    }
});
```

## Cycle de vie d'une application (suite et fin)

- Les applications NativeScript fournissent les événements sur les activités Android suivants :

- *ActivityCreated, activityDestroyed, activityStarted, activityPaused, activityResumed, activityStopped, saveActivityState, activityResultactivityBackPressed*

```
const application = require("tns-core-modules/application");

if (application.android) {
    application.android.on(application.AndroidApplication.activityCreatedEvent, function (args) {
        console.log("Event: "+args.eventName+, Activity: "+args.activity+, Bundle: "+args.bundle);
    });
}

application.android.on(application.AndroidApplication.activityDestroyedEvent, function (args) {
    console.log("Event: "+args.eventName+, Activity: "+args.activity);
});

application.android.on(application.AndroidApplication.activityStartedEvent, function (args) {
    console.log("Event: "+args.eventName+, Activity: "+args.activity);
});
}

application.run({ moduleName: "app-root" });
```

## Liaison de données

- La liaison de données vise à connecter l'interface utilisateur avec les données et à propager les modifications réalisées par l'utilisateur sur les données et vice versa
- Flux de données
  - Mono-directionnel (par défaut)
    - La propriété cible est mise à jour lorsque la donnée source change
    - l'interface utilisateur ne modifie pas la donnée source
  - Bi-directionnel
    - Les changements peuvent s'effectuer dans les deux sens
      - Source → cible
      - Cible → source
    - Un changement de valeur dans l'interface utilisateur entraîne un changement de valeur de la donnée source

## Liaison de données (suite)

- Principe de fonctionnement
  - Les objets cibles étendent la classe *Bindable*
    - Tous les éléments d'interface utilisateur héritent de cette classe
  - Bi-directionnel
    - la propriété cible doit être une propriété de dépendance
  - Mono-directionnel
    - Une propriété simple est suffisante
  - Les objets de données doivent générer un événement *propertyChange* pour notifier les éléments intéressés par les changements

## Liaison de données : exemples

- Liaison bi-directionnelle

```
// création de l'objet source de données possédant une propriété textSource
const fromObject = require("tns-core-modules/data/observable").fromObject;
const source = fromObject({
    textSource: "Text set via twoWay binding"
});

// création de éléments cibles
// create the TextField
const textField = require("tns-core-modules/ui/text-field").TextField;
const targetTextField = new TextField();

// create the Label
const label = require("tns-core-modules/ui/label").Label;
const targetLabel = new Label();

// binding the TextField
const textFieldBindingOptions = {
    sourceProperty: "textSource",
    targetProperty: "text",
    twoWay: true
};
targetTextField.bind(textFieldBindingOptions, source);

// binding the Label
const labelBindingOptions = {
    sourceProperty: "textSource",
    targetProperty: "text",
    twoWay: false
};
targetLabel.bind(labelBindingOptions, source);
```

## Liaison de données : exemples (suite et fin)

- Liaison avec un événement dans un fichier XML

```
<Page xmlns="http://schemas.nativescript.org/tns.xsd">
    <StackLayout>
        <Button text="Test Button For Binding" tap="{{ onTap }}" />
    </StackLayout>
</Page>
```

```
source.set("onTap", function(eventData) {
    console.log("button is tapped!");
});
page.bindingContext = source;
```

- Liaison avec un objet

```
const fromObject = require("tns-core-modules/data/observable").fromObject;

function onNavigatingTo(args) {
    const list = [];
    for (let i = 0; i < 5; i++) {
        list.push(new Date());
    }
    const source = fromObject({
        items: list
    });
    source.set("items", list);
    const page = args.object;
    page.bindingContext = source;
}
```

```
<Page navigatingTo="onNavigatingTo"
    xmlns="http://schemas.nativescript.org/tns.xsd">
    <StackLayout>
        <ListView items="{{ items }}" height="200">
            <ListView.itemTemplate>
                <Label text="{{ $value }}" />
            </ListView.itemTemplate>
        </ListView>
    </StackLayout>
</Page>
```

## Événements

- Les événements sont générés pour notifier une action spécifique
  - action utilisateur
    - pression sur un bouton, ...
  - action liée au programme
    - Fin de téléchargement d'une image, ...
- Exemple d'événement utilisateur

```
const Button = require("tns-core-modules/ui/button").Button;
const testButton = new Button();
testButton.text = "Test";

let onTap = function(args) {
    console.log("Hello World!");
};

// Adding a listener with the short syntax
testButton.on(buttonModule.Button.tapEvent, onTap, this);

// Adding a listener with the full syntax
testButton.addEventListener(buttonModule.Button.tapEvent, onTap, this);
```

## Événements (suite et fin)

- Souscription à un événement dans un fichier d'interface XML

```
<!-- main-page.xml -->
<Page>
    <StackLayout>
        <Label touch="onTouch" />
    </StackLayout>
</Page>
```

```
// main-page.js
function onTouch(args) {
    console.log("Touch arguments: ", args);
}
exports.onTap = onTap;
```

## Utilisation de greffons

- Les greffons NativeScript permettent de développer rapidement des applications mobiles en utilisant du code Javascript/TypeScript existant
- Les greffons peuvent permettre de faire le lien avec l'API native du système cible
- Il existe des dépôts de greffons
  - Maché officiel: <https://market.nativescript.org>
  - <https://www.nsplugins.com/>
- Les greffons s'installent via la commande `tns`
- Exemple d'utilisation d'un greffon

```
tns plugin add nativescript-camera
```

```
const cameraModule = require("nativescript-camera"); // Requiring the plugin module
camera.requestPermissions();
```

## Accès aux APIs natives depuis Javascript

- NativeScript fournit une API Javascript/TypeScript reflétant l'API native Android, et une autre reflétant celle d'IOS
- Les APIs sont semblables à celles des natives
- Elles sont définies dans le contexte global
- Pour accéder à une classe donnée, il faut préciser son paquetage
- Exemple

```
const javaLangPkg = java.lang;
const androidPkg = android;
const androidViewPkg = android.view;

// access classes from inside the packages later on
const View = androidViewPkg.View;
// or
const View = android.view.View;

const Object = javaLangPkg.Object; // === java.lang.Object;
```

## Accès aux APIs natives depuis Javascript : Exemple

```
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout linLayout = new LinearLayout(this);
        linLayout.setOrientation(LinearLayout.VERTICAL);
        LayoutParams linLayoutParam = new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT);
        setContentView(linLayout, linLayoutParam);

        LayoutParams lpView = new LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);

        TextView tv = new TextView(this);
        tv.setText("TextView");
        tv.setLayoutParams(lpView);
        linLayout.addView(tv);

        Button btn = new Button(this);
        btn.setText("Button");
        linLayout.addView(btn, lpView);

        LinearLayout.LayoutParams leftMarginParams = new LinearLayout.LayoutParams(
            LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        leftMarginParams.leftMargin = 50;

        Button btn1 = new Button(this);
        btn1.setText("Button1");
        linLayout.addView(btn1, leftMarginParams);

        LinearLayout.LayoutParams rightGravityParams = new LinearLayout.LayoutParams(
            LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        rightGravityParams.gravity = Gravity.RIGHT;

        Button btn2 = new Button(this);
        btn2.setText("Button2");
        linLayout.addView(btn2, rightGravityParams);
    }
}
```

## Accès aux APIs natives depuis Javascript : Exemple

```
const LinearLayout = android.widget.LinearLayout;
const LayoutParams = android.widget.LinearLayout.LayoutParams;
const TextView = android.widget.TextView;
const Button = android.widget.Button;
const Gravity = android.view.Gravity;

const MainActivity = android.app.Activity.extend("my.application.name.MainActivity", {
    onCreate: function (savedInstanceState) {
        super.onCreate(savedInstanceState);
        let linLayout = new LinearLayout(this);
        linLayout.setOrientation(LinearLayout.VERTICAL);
        let linLayoutParam = new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT);
        setContentView(linLayout, linLayoutParam);

        let lpView = new LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        tv.setText("TextView");
        tv.setLayoutParams(lpView);
        linLayout.addView(tv);

        let btn = new Button(this);
        btn.setText("Button");
        linLayout.addView(btn, lpView);
        let leftMarginParams = new LinearLayout.LayoutParams(
            LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        leftMarginParams.leftMargin = 50;

        let btn1 = new Button(this);
        btn1.setText("Button1");
        linLayout.addView(btn1, leftMarginParams);
        let rightGravityParams = new LinearLayout.LayoutParams(
            LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        rightGravityParams.gravity = Gravity.RIGHT;

        let btn2 = new Button(this);
        btn2.setText("Button2");
        linLayout.addView(btn2, rightGravityParams);
    }
});
```

# React Native

## Présentation de ReactNative

- ReactNative est un cadre de conception et un ensemble d'outils permettant de produire des applications mobiles multi-plateformes natives
- Il entre dans la catégorie « JIT-compiled » comme NativeScript
- ReactNative et NativeScript adoptent le même principe de fonctionnement
  - Écriture de l'application dans un langage et avec des composants graphiques indépendants des plateformes cibles
    - Ici en JSX (et pas en Javascript ou TypeScript)
    - Syntaxe permettant d'embarquer du XML dans du Javascript
  - Génération d'un code Javascript qui sera exécuté par la machine virtuelle Chrome V8 sur Android, et JavaScriptCore sur IOS

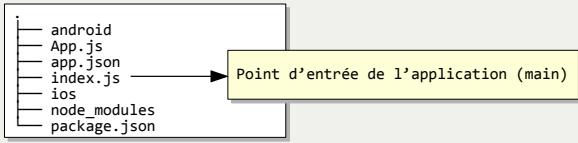
## ReactNative-CLI

- ReactNative s'installe via *npm*

```
npm install -g react-native-cli
```
- L'interface de ligne de commande de ReactNative permet
  - de créer une application
  - de compiler les sources
  - de générer l'APK
  - et d'installer et exécuter l'application sur un terminal (ou émulateur)

```
react-native init myApp  
cd myApp  
react-native run android
```

## Application ReactNative

- Arborescence des répertoires d'une application ReactNative
- Une application ReactNative est conçues comme un arbre de composants imbriqués

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

export default class HelloWorldApp extends Component {
  render() {
    return (
      <View>
        <Text>Hello world!</Text>
      </View>
    );
  }
}
```

## Spécialisation des composants

- Les composants peuvent être spécialisés lors de leur création (avec *props*) ou en cours d'exécution (avec *state*)
- *this.props*
  - Défini à l'instanciation par le composant parent
  - Sert à définir une propriété variable d'un composant à l'autre
  - Immuables en cours d'exécution
- *this.state*
  - Initialisé dans le constructeur du composant
  - Représente l'état du composant
  - L'état peut être changé en cours d'exécution avec la méthode *setState()*

## Spécialisation des composants : exemple d'utilisation de *props*

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Greeting extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Text>Hello {this.props.name}!</Text>
      </View>
    );
  }
}

export default class LotsOfGreetings extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Greeting name='Rexxar' />
        <Greeting name='Jaina' />
        <Greeting name='Valeera' />
      </View>
    );
  }
}

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => LotsOfGreetings);
```

## Spécialisation des composants : exemple d'utilisation de *state*

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Blink extends Component {
  constructor(props) {
    super(props);
    this.state = { isShowingText: true };

    // Toggle the state every second
    setInterval(() => (
      this.setState(previousState => (
        { isShowingText: !previousState.isShowingText }
      ))
    ), 1000);
  }

  render() {
    if (!this.state.isShowingText) { return null; }
    return (
      <Text>{this.props.text}</Text>
    );
  }
}

export default class BlinkApp extends Component {
  render() {
    return (
      <View> <Blink text='Hello' /> </View>
    );
  }
}

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => BlinkApp);
```

## Gestion des touches

```
import React, { Component } from 'react';
import { Alert, AppRegistry, Button, StyleSheet, View } from 'react-native';

export default class ButtonBasics extends Component {
  _onPressButton() {
    Alert.alert('You tapped the button!')
  }

  render() {
    return (
      <View style={styles.container}>
        <View style={styles.buttonContainer}>
          <Button onPress={this._onPressButton} title="Button 1" />
        </View>
        <View style={styles.buttonContainer}>
          <Button onPress={this._onPressButton} title="Button2" color="#841584"/>
        </View>
      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: { flex: 1, justifyContent: 'center' },
  buttonContainer: { margin: 20 },
  alternativeLayoutButtonContainer: { margin: 20, flexDirection: 'row', justifyContent: 'space-between' }
});

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => ButtonBasics);
```

## Gestion de la saisie de données

```
import React, { Component } from 'react';
import { AppRegistry, Text, TextInput, View } from 'react-native';

export default class PizzaTranslator extends Component {
  constructor(props) {
    super(props);
    this.state = {text: ''};
  }

  render() {
    return (
      <View style={{padding: 10}}>
        <TextInput
          style={{height: 40}}
          placeholder="Type here to translate!"
          onChangeText={(text) => this.setState({text})}
        />
        <Text style={{padding: 10, fontSize: 42}}>
          {this.state.text.split(' ').map((word) => word && ' '.join(' '))
        </Text>
      </View>
    );
  }
}

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => PizzaTranslator);
```

## Greffons

- Les greffons ReactNative sont à installer avec la commande `npm`
- Quelques exemples de greffons
  - `react-native-navigation` : navigation entre les différentes pages composant l'application
  - `react-native-contacts` : accès à la liste de contacts
  - `react-native-network-info` : informations sur la connectivité réseau
  - `react-native-permissions` : demande de permissions
  - `react-native-location-manager` : localisation

## Navigation dans une application

- La navigation entre les différents écrans composant une application s'effectue à l'aide du greffon `react-native-navigation`
- Exemple d'utilisation de ce greffon

```
import {
  createStackNavigator,
} from 'react-navigation';

const App = createStackNavigator({
  Home: { screen: HomeScreen },
  Help: { screen: HelpScreen },
});

export default App;
```

```
class HomeScreen extends React.Component {
  static navigationOptions = {
    title: 'Welcome',
  };
  render() {
    const { navigate } = this.props.navigation;
    return (
      <Button
        title="Help !"
        onPress={() =>
          Navigate('Help', { lang: 'FR' })
        }
      />
    );
  }
}
```