MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ ABDELHAMID IBN BADIS - MOSTAGANEM

Faculté des Sciences Exactes et d'Informatique

Département de Mathématiques et Informatique

Filière : Informatique

Polycopié de Cours
Applications Mobiles (AM)

Ce cours est destiné aux étudiants de 3^{iéme} Année LMD
Filière Informatique

Préparé par :

MECHAOUI Moulay Driss

Année Universitaire 2019-2020

Sommaire

Chapitre III: Les composants d'une application Android

Chapitre IV: Les interfaces graphiques Android

Chapitre V: Interactions entre les composants Android

Chapitre VI: Interfaces graphique avancées

Chapitre VII: La persistance des données dans Android

Chapitre I :

Introduction aux approches de développement mobiles

1 Introduction

Alors que le monde évolue vers des appareils mobiles plus connectés, l'utilisation des applications mobiles gagne en popularité. De nos jours, les appareils mobiles sont équipés d'une multitude d'applications pouvant accomplir une variété de tâches différentes. En effet, ils permettent aux utilisateurs de manipuler des applications puissantes qui profitent de la disponibilité croissante du réseau de communication intégré et de meilleures capacités d'échange de données. Le développement d'une application mobile efficace est devenu actuellement un enjeu important pour les entreprises afin d'étaler leurs services ou leurs produits, et d'établir une liaison directe avec les clients, et aussi pour le grand public afin de communiquer et collaborer avec les membres de la famille, les amis et collègues de travail.

2 Les systèmes d'exploitation mobiles

Nous présentons brièvement dans cette section les systèmes d'exploitation mobiles, leurs avantages et inconvénients. Tout d'abord, *un système d'exploitation mobile* est un système conçu pour fonctionner sur un dispositif mobile. Ce genre de système d'exploitation se focalise sur la gestion de : la connectivité sans fil, les différents types d'interface, la navigation internet et les appels téléphoniques et bien sur, la gestion de mémoire embarquée et des processus.

2.1 iOS

C'est un système d'exploitation mobile développé par Apple pour ses appareils mobiles (iPhone, iPad, iPod). C'est la version réduite de Mac OS X utilisé sur les ordinateurs de bureau de la marque Apple. La plateforme d'Apple est la pionnière et la plus populaire des plateformes mobiles, elle a lancée le marché des *smartphones* avec l'introduction d'*iPhone* en 2007. iOS est intuitive, ergonomique, facile d'emploi et irréprochable au niveau de la conception.

Toutefois, iOS est refermée sur lui-même, il ne laisse que très peu de place à la créativité de l'utilisateur. Il ne permet pas de vendre ou de distribuer des applications que sur son magasin (App Store). Il garde le contrôle total sur les applications distribuées, puisqu'il oblige la validation et l'enregistrement de développeur ainsi que le matériel qu'il utilise auprès d'Apple.

2.2 Windows Phone

C'est un système d'exploitation mobile conçu et développé par Microsoft. Ce système est caractérisé par l'interface utilisateur similaire à l'interface Metro utilisée sur le système d'exploitation Windows 8. Metro est une interface simple, personnalisable, mais très réactive. Il est considéré comme le système d'exploitation mobile le plus adapté aux utilisateurs néophytes (inexpérimentés).

Toutefois, Window Phone n'est pas open-source et ses outils de développement d'applications sont payants ce qui l'a rend peu populaire. Il souffre également d'une concurrence accrue avec iOS et Android.

2.3 Android

C'est est un système d'exploitation open source pour terminaux mobiles (Smartphones, Tablettes, liseuses, etc.), fondé à partir du kernel de Linux et de la plateforme de programmation Java. Il a été créé à la base par la startup Android et rachetée par la suite par Google en 2005. Pour la promotion de ce système, et faire face à la domination de iPhone sur le marché mondiale, Google a fédéré autour de lui plusieurs partenaires (des sociétés technologiques, des fabricants de périphériques, des opérateurs sans fil, etc.) réunis au sein de l'Open Handset Alliance (OHA) dont le but est de développer des standards ouverts pour les appareils mobiles.

Actuellement, c'est le système d'exploitation mobile le plus utilisé à travers le monde. Leurs versions peuvent être adaptées et personnalisée en fonction de la marque de l'appareil et/ou son opérateur de base. Ceci crée une grande concurrence sur le prix, le design, les caractéristiques techniques, et la qualité. En plus, il bénéficie d'une énorme communauté de développeurs et d'une documentation bien fournie. Il touche pratiquement tous les objets connectés (Téléviseur, montre, climatiseur, lunette, etc.)

Toutefois, cette fragmentation énorme des différentes versions pose un problème de teste et de mise à jour. A cause de ça, Android se retrouve avec un système d'exploitation moins stable et moins conçu que l'iOS.

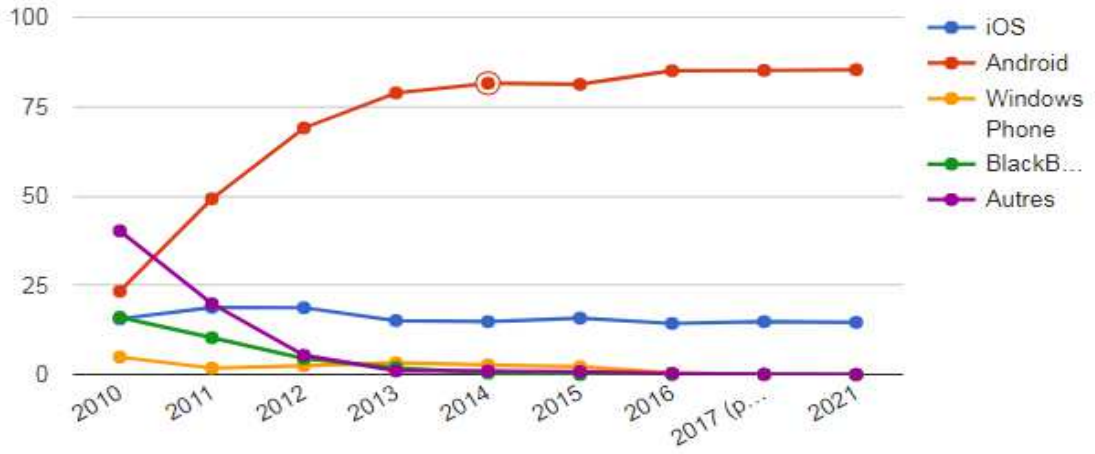


Figure1. Part du marché mondiale des OS mobiles (%) [1]

Il existe d'autres systèmes d'exploitation mobiles dans le marché tels que *Synbian OS* de Nokia, *Blackberry OS* de RIM (Research In Motion), et *Bada* de Samsung. Mais, ces systèmes ont du mal à rivaliser avec Android et iOS sur le marché mondial.

3 Le développement d'applications mobiles

3.1 Application mobile

Une application mobile est un programme conçu pour fonctionner sur les dispositifs mobiles tels que smartphones, tablettes, PDA, etc. Elle est adaptée aux divers environnements techniques des dispositifs mobiles et à leurs contraintes et options ergonomiques. Elle offre un accès efficace et confortable à des sites ou services accessibles par ailleurs en versions mobile ou web. Actuellement, les plateformes de distribution (Stores) les plus populaires pour les applications mobiles sont App Store, Google Play, et Windows Phone Store.

3.2 Les contraintes de développements

Le développement d'une application mobile repose généralement sur les contraintes suivantes : ***(i) Les caractéristiques des dispositifs*** *mobiles* telles que la consommation d'énergie, capacité de calcul, l'intermittence des connexions, la limite de la bande passante, capacité de stockage, et l'hétérogénéité de leurs systèmes d'exploitation, et les frameworks relatives à leurs spécificités matérielles. ***(ii) Le concept du marché***, le processus du développement doit réduire au maximum le « *Time to Market* (TTM) » qui est le temps qu'il faut entre la conception d'un produit et sa mise en vente. Le TTM temps est important dans

[1] http://www.zdnet.fr/actualites/chiffres-cles-les-os-pour-smartphones-39790245.htm

les industries où les produits sont démodés rapidement. Ce temps est important puisque les concepteurs des applications mobiles doivent cibler plusieurs systèmes (Android, iOS, Windows Phone, BlackBerry, ect) dans le marché mobiles, ce derniers est très diversifier comme le montre la Figure1. La contrainte la plus importante est ***(iii) le choix de la plateforme*** de développement. Ce choix dépond des caractéristiques de l'application développée, de l'accès aux fonctionnalités du dispositif mobile cible, et la fragmentation importante du parc mobile (taille d'écrans, types et versions des systèmes d'exploitation, performances matérielles…). Alors le développeur doit connaître les avantages et les limites des différentes plateformes de développement mobile par rapport aux besoins et aux objectifs de leur l'application avant de faire le choix. D'autres contraintes peuvent être citées sous formes de questions : Quel est le public cible de l'application mobile ? À quoi va-t-elle servir ? Par quel dispositif mobile et sur quel système d'exploitation ? Est-ce que l'application à besoin d'intégrer des fonctionnalités natives ?

4 Les approches de développement mobile

Une application mobile est le moyen le plus efficace pour mettre une entreprise ou un service à la disposition des utilisateurs sur leurs dispositifs mobiles, mais avec différents systèmes d'exploitation et dispositifs mobiles (Android, iOS, et WindowsPhone) qui nécessitent des codes source différents pour fonctionner, viser un public mobile n'est pas toujours aussi simple que de créer une version mobile d'un site. Alors le choix de l'approche de développement est essentiel pour toucher un grand public d'utilisateurs ainsi que de dispositifs mobiles.

4.1 Le développement générique (Web)

Un site web mobile est une application mobile développée en HTML (HyperText Markup Language) et accessible via le navigateur web du dispositif mobile (smartphone et tablette). C'est la manière la plus simple pour une entreprise pour être présent sur un mobile. Un site web mobile n à pas besoin de télécharger l'application sur le mobile, et il est accessible par tous les dispositifs mobiles sans exception. Toutefois, la plus part des applications web ne prennent pas en charge les différents modèles et systèmes d'exploitation des dispositifs mobiles. Ces applications ne sont pas toujours ergonomiques, par exemple, quelques applications web ont une mauvaise résolution sur certains mobiles, et une interface

inadaptée sur la majorité des écrans. Afin de proposer un site web mobile visible, navigable, et avec un contenu optimisé ; le développeur peut opter pour une version dédiée de site ou une version responsive design.

1) ***Le site dédié*** : est un projet qui consiste à développer plusieurs versions de même site selon le dispositif mobile visé, par exemple, une version pour le site principal, une version pour smartphone et une autre pour tablette. Le serveur de site utilise un test initial pour détecter le type de dispositif de connexion et renvoyer vers une adresse web dédiée (***m.monsite.dz***).
2) ***Le Responsive Web Design*** : est une approche de conception web qui s'adapte à tout type d'appareil (smartphones, tablettes, PC de bureau) de façon transparente pour l'utilisateur. Elle propose une interface d'affichage et de navigation optimale qui utilise le HTML et CSS (Cascading Style Sheets) pour redimensionner, agrandir ou déplacer le contenu de la page web afin de le rendre agréable à l'écran. Cette approche est fondée sur trois principes essentiels:
    - Des grilles fluides, utilisant un dimensionnement relatif
    - Des images fluides, à travers le redimensionnement
    - Media Queries : est une fonctionnalité de langage CSS qui permet de mettre en page les sites web.

Cependant, les performances de ces applications sont modestes puisque le temps de leurs chargements sur le mobile est relativement long. De plus, Il est difficile d'éviter les limites ergonomiques et des performances des navigateurs web mobiles. Cette approche n'offre pas d'accès aux fonctionnalités "natives" des systèmes et nécessite une connexion Internet.

4.2 Le développement natif

C'est une approche traditionnel pour développer une application en générale, cette approche utilise le langage natif de dispositif mobile pour implémenter une application mobile spécifique au système d'exploitation visé (*Java* pour Android, *Objective-C* pour iPhone, *C#* pour Windows Phone, etc.). En principe, une application native Android ne pourra pas fonctionner sur un téléphone iPhone et vis versa. Les applications natives permettent l'utilisation de la mémoire de l'appareil mobile ainsi que toutes les fonctions associées à son système d'exploitation (GPS, accéléromètre, appareil photo, etc.). Ce type

d'applications est distribué à travers les magasins (stores) d'applications propres à chaque plateforme (*Play Store de* Google, *App Store* d'Apple, et *Windows Phone Store* de Windows Phone) pour les télécharger et les installer.

Ces applications possèdent plusieurs avantages tels que la performance, simplicité d'implémentation, flexibilité d'exécution, et la possibilité d'accès aux applications natives du dispositif mobile (contact, agenda, etc.), et aux ressources matérielles dont il dispose (GPS, accéléromètre, le gyroscope, etc.). La conformité de ces applications aux ergonomies particuliers de chaque système d'exploitation leurs donnent la meilleure expérience utilisateur possible.

Système d'Exploitation	IDE	Langage	Store	Open Source
Android	Android Studio	Java	Google Play	Oui
iOS	Xcode	Objective-C	App Store	Non
Windows Phone	MicroSoft Visual Studio	C#	Windows Phone Store	Non

Table1. Comparaison entre les plateformes mobiles dominantes actuellement.

L'inconvénient de cette approche c'est qu'il faut implémenter une version dédiée à chaque système d'exploitation, du coup, le cout de développement sera élevé. En plus, les mises à jour nécessitent un coût de développement supplémentaire multiplié par le nombre de plate-formes cibles.

4.3 Le Cross-platform (multiplateformes) développement

C'est une technologie qui permet de générer plusieurs applications à destination de différents systèmes d'exploitation mobiles à partir d'un seul code. Cette technologie est très bénéfique pour les entreprises, puisqu'elle évite :

- Le recrutement de développeurs spécialisés pour chaque plate-forme,
- La création de la même application pour chacune des plateformes.
- La maintenance et la mise à jour de ces différentes versions qui peuvent s'avérer complexes, coûteuses et longues.

Le développement « Cross-platform » permet de concevoir deux types d'applications mobiles: les applications hybrides et les applications natives générées.

4.3.1 Le développement hybride orienté Web

Les applications hybrides sont constituées de 80% de technologies Web comme HTML, CSS, et JavaScript combinée avec 20% de fonctionnalités natives des systèmes d'exploitation mobiles. Ces applications permettent de réduire les coûts et délais de développement, puisqu'elles n'ont pas besoin de la réécriture de code à chaque changement de plate-forme. Contrairement aux applications web qui sont consultables à travers un navigateur, les applications hybrides sont téléchargées (App Store, Android Market, etc.) et installées sur le dispositif mobile. Plusieurs framework permettent la création et le déploiement des applications hybrides, on peut citer, *Apache Cordova*[2], *Phonegap*[3], et *Titanium*[4] :

Apache Cordova : c'est un framework de développement mobile open-source, proposé pour simplifier la création d'applications mobiles pour différentes plateformes en utilisant les technologies Web comme HTML5, CSS3, et JavaScript. Apache Cordova encapsule le code dans un WebView (fenêtre de navigateur spécial) et génère une application native qui sert de conteneur pour lancer la WebView. Il utilise le JavaScript pour accéder aux fonctionnalités natives matériels (GPS, accéléromètres, gyromètres, etc.) et logiciels (contacts, photos, notifications...) du mobile.

[2] https://cordova.apache.org/
[3] https://phonegap.com/
[4] http://www.appcelerator.org/

Les applications hybrides sont utilisées souvent pour « porter » rapidement et avec un faible coût, un site web sous forme d'application téléchargeable disponible via les stores (App Store, Google Store) et opérationnelles sur plusieurs systèmes d'exploitations mobiles. Cependant, la performance et la qualité de ces applications sont très limitées surtout pour les projets complexes. Ces application n'offre pas un accès à l'ensemble des fonctionnalités natives, et elles ne sont accessibles, souvent, que sur Android et IPhone, et refusées sur d'autres plateformes.

4.3.2 Applications natives générées

Les frameworks de développement de ce type d'applications natives permettent de compiler un seul code source d'application en code natif qui s'exécutera sur différents systèmes d'exploitation. Les frameworks tels que *Xamarin, Appcelerator* et *Adobe Air* génèrent des applications basées sur un langage commun (***C#*** pour Xamarin, ***Javascript*** pour Appcelerator, et ***AS3*** pour Adobe Air) et des outils uniques afin de produire de véritables applications natives.

Xamarin[5] : c'est un framework qui permet le développement d'applications natives pour différentes plateformes (Android, iOS, et Windows Phone) en utilisant le langage C# (dite C sharp). Le développeur crée une base de code commune contenant : le logique métier, le stockage en base de données, les appels réseaux, les éléments d'interface communs. Ensuite, plateforme cible créée un projet qui contient les composants propres à chaque SDK cible et l'interface graphique. Xamarin donne aussi la possibilité de profiter des spécificités propres à chaque plateforme sans réduire l'expérience utilisateur. Une fois le code source réalisé, Xamarin Studio (IDE de Xamarin) compile et génère trois fichiers exécutables (binaire natif) pour les terminaux Android (***.apk***), *iOS* (***.ipa***) et Windows Phone (***.appx***).

Cependant, Xamarin a aussi des limites, par exemple, un développeur Java devrait avoir une connaissance préalable de langage C# avant d'utiliser Xamarin. En plus, il doit apprendre les subtilités propres à chaque plate-forme cible afin de s'adapter aux éléments natifs de leur SDK. Cela peut consommer plus de temps que de passer par du développement natif.

[5] *https://xamarin.com/*

Ces inconvénients ne sont pas propres uniquement à Xamarin, mais aussi aux autres frameworks de développement des applications natives générées. Ajouté à cela, la difficulté de maintenir et de faire évaluer ces applications à cause des différentes étapes de compilation, et de contenu peu natif du code source.

5 Conclusion

Dans ce chapitre nous avons présenté les différentes approches de développement des applications mobiles. Chaque approche propose des avantages et souffre de lacunes. Le choix d'une approche dépend généralement des contraintes (public ciblé, type de plateforme, fonctionnalités requises) de l'application à développer.

Questions

- Comment peut-on choisir l'approche la plus adéquate pour développer une application mobile ?
- Expliquer l'importance de la notion TTM (Time to Market) pour le processus de développement d'une application mobile
- Pourquoi la conception des applications de iPhone est irréprochable contrairement à celle d'Android ?

Chapitre II :

Les composants de la plate-forme Android

1 Introduction

Android est une plate-forme open-source innovante qui support les dernières technologies de téléphonie (écran, accéléromètre, GPS, appareil photo, etc.). Ce chapitre est une introduction aux plates-formes, aux outils et aux différents environnements de développement des applications mobiles.

2 Composants de la plate-forme Android

La plateforme Android est n'est pas conçue uniquement pour des appareils mobiles, elle offre d'autres possibilités d'utilisation pour supporter les tablettes, les ordinateurs portables, les téléviseurs, les lunettes, etc. La plate-forme Android est composée de différentes couches, la figure suivante illustre les principaux composants de la plate-forme Android.

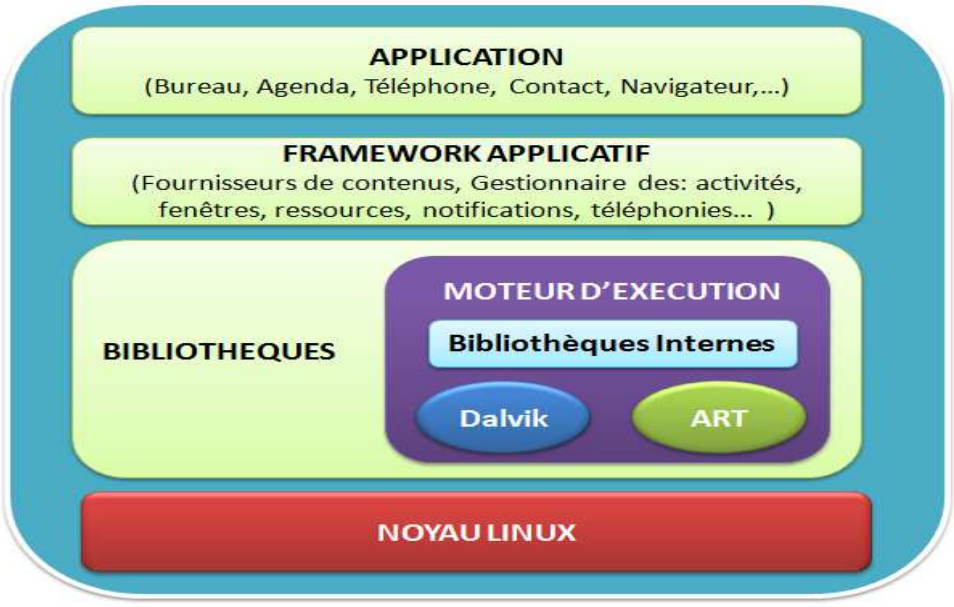


Figure2. Les composants de la plateforme Android

2.1 Application

Cette couche est visible et accessible à l'utilisateur, elle lui fournit un ensemble d'applications (implémentées en Java) offrant un accès aux fonctionnalités de base telles que le bureau, l'agenda, le navigateur, les SMS, le téléphone, les photos, etc.

2.2 Framework Applicatif

Cette couche est basée sur le principe de la réutilisation des composants et leur coopération. Elle est considérée comme un point d'entrée pour accéder aux librairies fournies par le système et les ressources matérielles. Elle contient tous les services auquel il sera possible

de faire appel pour récupérer des informations des couches inférieures. Parmi les services assurés par cette couche on peut citer :

- Fournisseur du Contenu : Il permet à une application d'accéder aux données d'une autre application (contact, agenda), ou de partager ses propres données (base de données, fichiers.)
- Gestionnaire d'Activité : il gère le cycle de vie de l'application en cours d'exécution.
- Gestionnaire de fenêtres : fournis et gère toutes les informations concernant l'écran.
- Gestionnaire de Resource: Il assure l'accès à toutes les ressources (image, fichiers, chaîne de caractères, layout).
- Gestionnaire de Notification : le système d'information utilisateur, nommé notification (réception d'un SMS par exemple). Il permet aux applications d'afficher des alertes personnalisées dans la barre d'état.
- Gestionnaire de téléphonie : offre les informations et les services liés à la téléphonie.
- Système de vues : c'est un système interne qui permet la création des vues pour l'interface graphique d'une application en utilisant des boutons, listes, grilles, cases à cocher, et même un navigateur web embarqué.

2.3 Bibliothèques

Plusieurs composants de la plateforme Android font appel à un ensemble de bibliothèques C et C++. Ces bibliothèques sont accessibles à travers le framework applicatif. Parmi les bibliothèques disponibles dans Android :

- *Bibliothèque système C* : C'est une implémentation de la bibliothèque standard C (libc), c'est une version optimisée pour les systèmes Linux embarqués.
- *Bibliothèques multimédias* : Elles permettent à Android de supporter la lecture et l'enregistrement de plusieurs formats vidéo, audio, les fichiers images statiques tels que MPEG4, MP3, JPG, PNG, etc.
- *Freetype* : C'est une bibliothèque qui fournit un moteur de rendu de police de caractères. Elle est implémentée en C et permet d'obtenir un rendu bitmap des polices vectorielles.
- *SQLite* : C'est un système de gestion de base de données relationnelle embarqué disponible pour toutes les applications.

2.3.1 Moteur d'exécution (Android RunTime)

Android comprend un ensemble de bibliothèques qui fournit les fonctionnalités disponibles dans les bibliothèques noyau du langage de programmation Java.

Dalvik et ART sont des machines virtuelles qui permettent aux applications de s'exécuter sur un dispositif mobile Android, indépendamment des différences matérielles et quelque soit le modèle.

2.3.2 Dalvik

C'est une implémentation de machine virtuelle adaptée aux systèmes restreints en mémoire et en puissance de processeur, elle est conçue pour optimiser l'exécution multiple de machines virtuelles puisque chaque application s'exécute dans sa propre instance de la Dalvik VM.

Dalvik introduit le compilateur JIT (Just-*In-Time* ou bien *juste à temps* en français) à partir de la version Android 2.2 (Froyo), pour optimiser et améliorer les performances des applications.

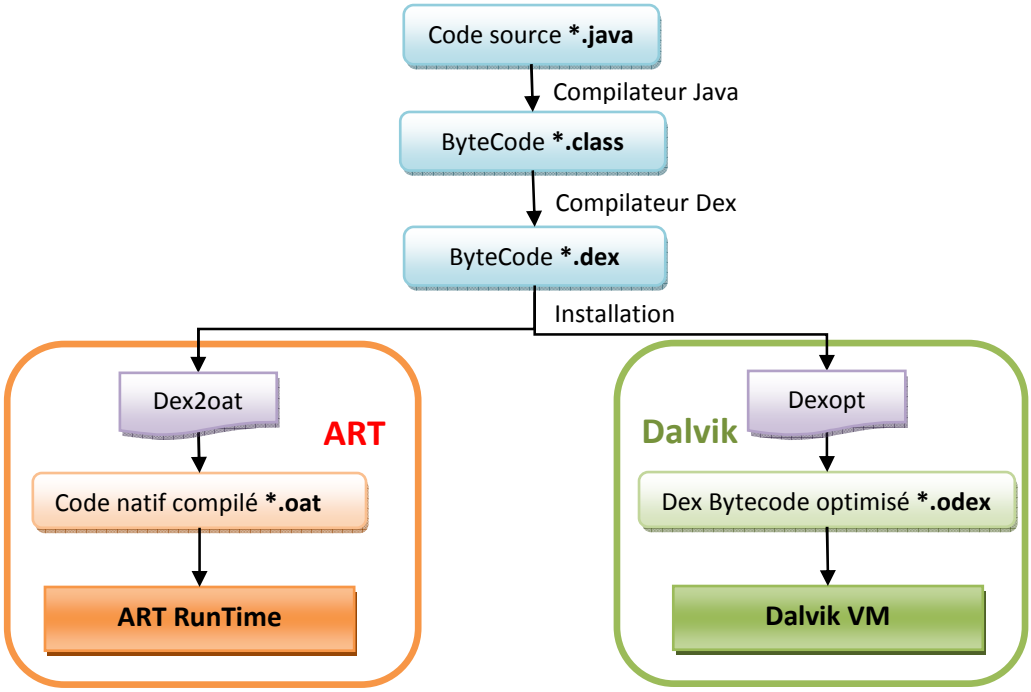


Figure3. Dalvik et ART

Chaque fois qu'une application est lancée, le bytecode est interprété ligne par ligne. Lorsqu'une une fonction est identifiée comme hotspot (un extrait du code qui s'exécute très fréquemment). Elle est directement compilée (à la volée) en code natif par JIT avec optimisation (odex). Dalvik utilise un outil appelé dexopt qui permet d'optimiser les fichiers .dex et de produire un fichier .odex similaire aux fichiers .dex d'origine, sauf qu'il utilisait des opcodes (*operation code : c'est une portion d'une instruction en langage machine qui spécifie l'opération à effectuer*) optimisés. Cela cause des pertes de temps et d'espace mémoire, mais par la suite, ce code compilé par JIT est exécuté directement (sans interprétation), ce qui augmente les performances.

2.3.3 ART

ART (Android RunTime) a été introduit dans Android 4.4 (kitkat) optionnellement, mais à partir d'Android 5.0 (Lollipop), il a complètement remplacé Dalvik. ART utilise un outil appelé dex2oat pour convertir les fichiers .dex en fichiers ELF[6] (Executable and Linkable Format) qui peuvent être exécutés nativement. ART utilise la compilation AOT (*Ahead of time*, à l'avance) anticipée, en compilant l'application lors de son installation, sans avoir besoin d'une interprétation ultérieur. ART permet ainsi d'augmenter les performances, et par conséquence d'augmenter la durée de vie de la batterie puisqu'il exécute le code machine de l'application directement (exécution native). En plus, réduit le temps de lancement des applications comme le code natif est directement exécuté. Cependant, Une application compilée par ART nécessite plus de temps pour la compilation lorsque l'application est installée et prend un peu plus d'espace interne pour stocker le code compilé.

En résumé, la différence majeure entre Dalvik et ART se situe sur le moment d'interprétation du code. Dalvik interprète le bytecode à la volée pour être exécuté (JIT : Just in Time). Alors que ART compile le bytecode en avance (au moment d'installation) une fois pour toutes, avant son utilisation (AOT : Ahead-Of-time).

2.4 Noyau linux

Android est fondé sur un noyau Linux (version 2.6), ce noyau permet de gérer divers services du système tels que la gestion des processus, des threads, de la mémoire, des drivers, de la sécurité, et du réseau. Il acte en tant qu'une couche d'abstraction entre le matériel et la pile logicielle.

La couche HAL (Hardware Abstraction Layer) est située entre les bibliothèques et le noyau linux, elle propose des interfaces implémentant les pilotes du noyau qui permettent de séparer la plateforme logique des interfaces matérielles. L'objectif de cette couche est d'assister le portage des librairies sur différents matériels, et d'éviter aux développeurs d'implémenter manuellement des codes particuliers à chaque périphérique.

[6] ELF : est un format de fichier binaire utilisé pour stocker de code compilé (objets, exécutables, bibliothèques de fonctions).

3 Environnement de développement

3.1 Android SDK

Le kit de développement Android (SDK : Standard Development Kit) fournit aux développeurs un environnement de travail pour implémenter, tester et débuguer des applications Android.

Au fur et à mesure que la plate-forme Android évolue et que de nouvelles versions d'Android sont publiées, chaque nouvelle version d'Android obtient un identifiant entier unique, appelé niveau API (API Level). Chaque version d'Android est composée de plusieurs noms, la dernière version actuellement:

- La version Android, comme ***Android 10***
- Un nom de code, tel que ***Q***
- Un niveau d'API correspondant, tel que ***API 29***

Version	Nom de code	Niveau API	Année
Android 10	Q	API 29	2019
Android 9.0	Pie	API 28	2018
Android 8.1	Oreo	API 27	2017
Android 7.1	Nougat	API 25	2016
Android 6.0	Marshmallow	API 23	2015
……	…….	……..	……
Android 1.5	Cupcake	API 3	2009

Table2. Quelques versions précédentes de Android

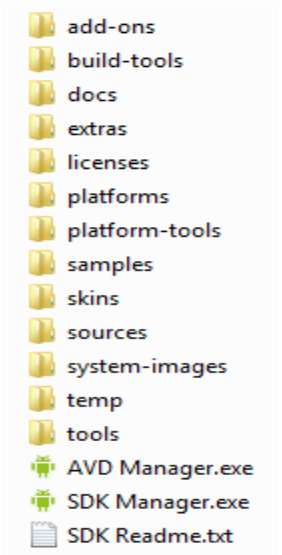
Chaque version d'Android est attribuée uniquement à un seul niveau d'API. Cependant, le nom du code Android peut correspondre à plusieurs versions et niveaux d'API.

Les applications Android doivent être conçues pour supporter plusieurs niveaux d'API Android, puisque que les utilisateurs installent des applications sur différentes versions d'Android.

3.2 SDK Manager :

Le SDK Manager permet de télécharger les plateformes, les outils, et de connaître la version du SDK installée et permet aussi d'installer de nouvelles lorsqu'elles deviennent disponibles.

- Android versions xx
- Google API versions xx
- Outils (tools et platform-tools)



3.3 AVD Manager

Cet outil sert à créer et gérer les terminaux virtuels (AVD) qui accueilleront les instances de l'émulateur. Il permet de configurer un AVD par l'attribution d'un nom, d'une version d'Android (disponible dans le SDK), d'une capacité de carte SD et d'une résolution d'écran.

3.4 Les Outils SDK

Adb (*Android Debug Bridge*) :

C'est un outil de ligne de commande basé sut le modèle client-serveur permettant de communiqué avec les dispositifs mobiles. Il est composé de trois composants:

1. Un démon exécuté par l'émulateur;
2. Un serveur assurant la gestion de communication entre le client et le démon qui s'exécute sur un émulateur ou un dispositif mobile réel.
3. Un client exécuté sur la machine de développement, et communique (via des lignes de commandes) avec le démon via le serveur;

adb est utilisé pour gérer une instance d'émulateur ou d'un dispositif Android réel. Il est accessibles à travers une ligne de commande (fenêtre d'invité de commande) pour :

- Transférer des fichiers (push / pull) : `C:\> adb push source destination`
- Installer une application (install) : `C:\> adb install NomDuFichierApk`
- Paramétrer le réseau (forward) : `C:\> adb forward <local> <remote>`

dx

C'est un outil qui permet la création du bytecode Android depuis un fichier .class. Il transforme les fichiers et/ou répertoires en exécutable Dalvik sous le format *.dex*, pour le faire fonctionner dans l'environnement Android. Il peut aussi restaurer le fichier *.class* en format lisible (*.java*).

apkbuilder

C'est un outil qui permet de compiler les sources d'une application Android pour constituer une archive (*.apk*) directement installable sous un terminal Android.

DDMS (Dalvik Debug Monitor Service)

C'est un outil de débogage qui permet aux développeurs d'interroger les processus actifs, d'examiner la pile et le tas, de surveiller et mettre en pause les threads actifs et explorer le système de fichier de n'importe quel dispositif Android connecté.

aapt (Android Assert Packaging Tool)

C'est un outil du SDK qui permet de visualiser, créer et mettre à jour les fichiers APK (ainsi que les fichiers zip et jar) d'une application. Il peut également compiler des ressources (le fichier AndroidManifest.xml et les fichiers XML) d'une activité dans des ressources binaires. C'est le constructeur basique des applications Android. Bien que les développeurs n'utilisent pas généralement *aapt* directement, les scripts de construction et les plugins IDE utilisent cet outil pour empaqueter le fichier apk qui constitue une application Android.

Emulateur

C'est un outil essentiel pour le test et le débogage d'une application Android. Afin de modéliser et de tester une application plus facilement, l'émulateur utilise les configurations de périphérique virtuel Android (AVD : Android Virtuel Device). Le AVD est une configuration d'émulateur qui permet de modéliser un dispositif réel en définissant les options matérielles et logicielles à émuler par l'émulateur Android. Une fois l'application exécutée sur l'émulateur, elle peut utiliser les services de la plate-forme Android pour faire appel à d'autres applications, accéder au réseau, lire des fichiers audio et vidéo, stocker et récupérer des données, informer l'utilisateur et afficher les transitions graphiques et les thèmes. Il existe plusieurs émulateurs Android tels que : Android Studio Emulator[7], AMIDuOS[8], Andy[9], ARChon[10], Genymotion[11], etc.

On peut bien sur utiliser carrément un dispositif réel avec une connexion usb, il suffit d'installer le pilot de dispositif réel sur le PC et de coucher l'option de *Débogage USB* dans le menu *option de développement* dans les paramètres de ce dispositif.

3.5 IDE et Plugins

Il existe plusieurs l'environnement de développement intégré (IDE) qui sont destinés généralement au développement d'applications mobile Android tels que : Eclipse, Android Studio, IntelliJ IDEA , NetBeans (avec plugin Android), etc.

IDE Eclipse

Eclipse est l'environnement de développement intégré (IDE) le plus populaire dans la communauté des développeurs Java. Il est constitué d'un espace de travail (workspace) de base et un système de plug-in extensible pour

[7] https://developer.android.com/studio/index.html
[8] http://www.amiduos.com/
[9] https://www.andyroid.net/
[10] https://github.com/vladikoff/chromeos-apk/blob/master/archon.md
[11] https://www.genymotion.com/#!/

personnaliser l'environnement. Eclipse est implémenté en Java et son objectif principale est de développer des applications en Java, mais il peut également être utilisé pour implémenter des applications dans d'autres langages de programmation à travers les plugins.

ADT plugin

Le plugin eclipse ADT (Android Development Tools) simplifie le développement Android. Il fournit un assistant de projet Android qui permet de créer rapidement de nouveaux projets, et de créer une interface utilisateur d'application. Il automatise et simplifie le processus de construction des applications Android, et intègre aussi les outils de développement comme l'émulateur et le convertisseur class-to-.dex. Il offre aussi un éditeur visuel qui évite de manipuler directement le XML, cet éditeur aide à écrire du code XML valide pour le manifeste Android (AndroidManifest.xml) et les fichiers de ressources.

Android Studio

C'est l'IDE officiel d'Android depuis 2014, c'est un environnement de développement et de programmation entièrement intégré. Il offre des outils de création d'applications plus rapides sur tous types d'appareils Android. Il permet de visualiser en temps réel les changements effectués sur une application, et on peut voir aussi voir aussi son effet sur différents appareils Android. Lorsqu' un nouveau projet est lancé avec Android studio, la structure du projet apparaît avec presque tous les fichiers du répertoire SDK. Android Studio utilise Gradle. C'est une boîte à outils de construction avancée, qui permet d'automatiser et de gérer le processus de construction (build), tout en définissant des configurations de construction personnalisées et flexibles. Chaque configuration de build définie son ensemble de code et de ressources, et réutilise des parties communes à toutes les versions d'une application.

AIDE

Est un environnement de développement intégré complet pour les dispositifs mobile. C'est une application Android ordinaire qui s'installe dans le dispositif mobile. L'application AIDE est fournie avec une version mobile du SDK Android, il n'est donc pas nécessaire d'installer quoi que ce soit d'autre. C'est un moyen facile d'apprendre le développement Android et livré avec des leçons intégrées et des tutoriels. AIDE permet de créer des applications à la volée. Bien que AIDE ne remplacera pas bientôt Android Studio, car il a certainement ses limites en termes de flux de travail, mais il permet de développer des applications simples (avec Java, C/C++, HTML5, CSS et JavaScript) ou de tester des idées directement sur votre propre appareil au lieu d'utiliser un émulateur. En plus, il intègre Dropbox et Git, et il est compatible avec Eclipse et Android Studio.

Flutter

Est un kit de développement de logiciels d'interface utilisateur open source créé par Google en 2017. Il est permet de développer des applications pour divers plateformes tells que Android, iOS, Windows, Mac, Linux, Google Fuchsia et même le web. L'objectif est de permettre aux développeurs de fournir des applications hautes performances qui semblent naturelles sur différentes plates-formes.

3.6 Langages

Depuis la création d'Android, JAVA était le langage officiel bien qu'il support d'autres langages tels que HTML, CSS, JavaScript et C/C++ en natif. Cependant en 2017, Google annonce que ***Kotlin*** devient officiellement le second langage de programmation supporté sur Android après Java.

Kotlin[12] est un langage de programmation orienté objet, avec typage statique qui peut s'exécuter sur une machine virtuelle Java et peut également être compilé en code source JavaScript ou utiliser l'infrastructure du compilateur LLVM (Low Level Virtual Machine). Il est basé sur l'interopérabilité, la sécurité, la clarté et le support des outils. Kotlin réduit considérablement la quantité de code passe-partout (boilerplate code), et cela signifié moins d'erreurs, et par conséquent moins de crashs.

Dart (dite Dash) est un langage de programmation web développé par Google dont le but est de remplacer JavaScript dans le développement web.

Dart permet le développement des applications côté serveur et même les applications mobiles (via l'API Flutter). Dart Native supporte différents types d'appareils (mobiles, PC, serveurs, etc.). Il inclut à la fois une machine virtuelle Dart avec compilation JIT (juste à temps) et un compilateur AOT (anticipé) pour créer du code machine.

[12] Kotlin, est une île près de St. Pétersbourg en Russie

4 Conclusion

Dans ce chapitre nous avons présenté les composants principaux de la plateforme Android. La figure suivante résume les étapes d'exécution d'une application Android du code source java jusqu'à son dépoilement sur le terminal mobile.

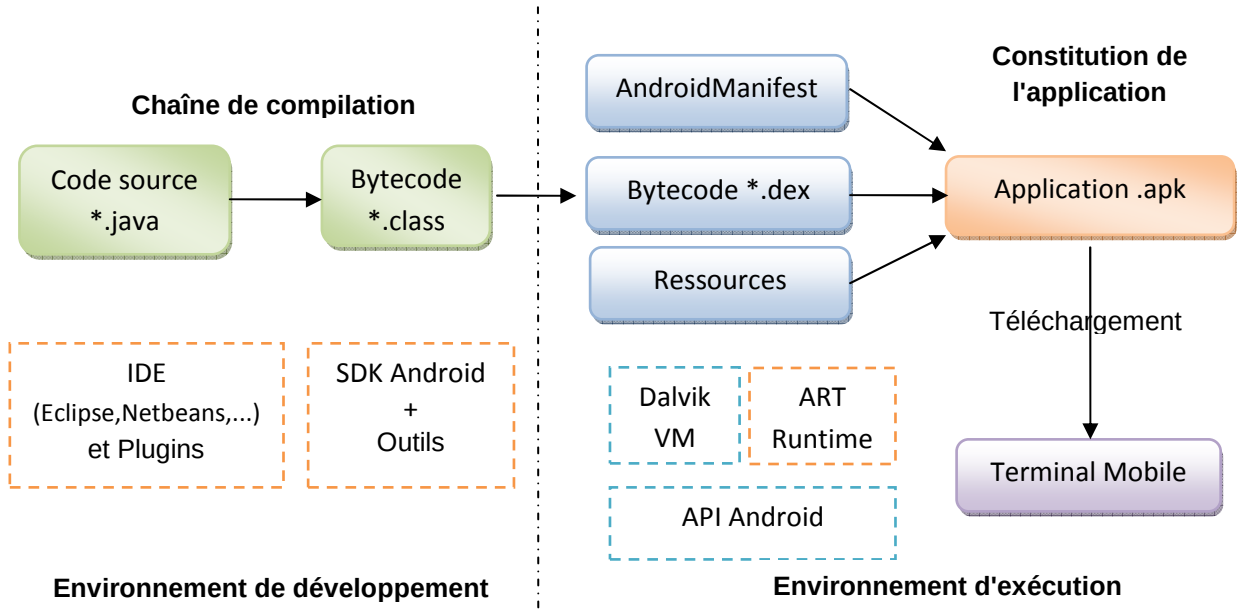


Figure4. Les étapes d'exécution d'une application Android

Questions

- Tracer un tableau comparatif entre Dalvik et ART en se basant sur les critères suivants : temps installation, compilation, temps d'exécution, espace de stockage nécessiare.
- Quel est l'outil d'Android SDK qui permet de faire la liaison entre un IDE et un dispositif mobile ?

Chapitre III :

Les composants d'une application Android

1 Introduction

Une application Android est un montage de composants (activités, vues, ressources) liés par un fichier de configuration (manifeste). Ce chapitre présente les différents composants d'une application Android et leurs interactions.

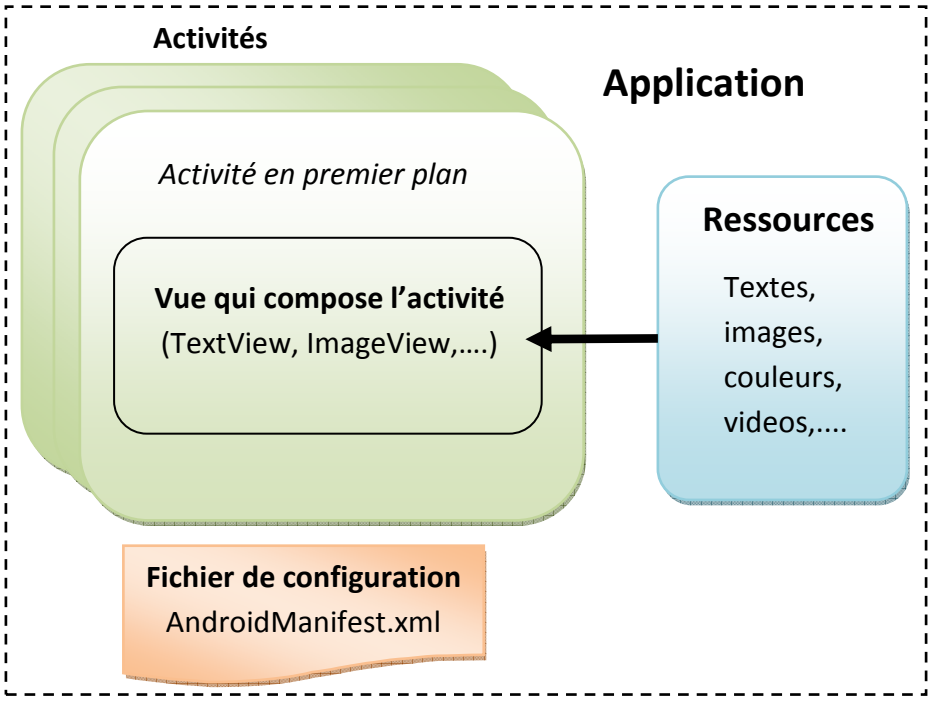


Figure5. Composition d'une application Android

2 Activité

Une activité est l'unité de base pour une application Android. Elle possède le code (java ou kotlin) de l'application, et garantit la gestion de l'ensemble des vues et ressources. Chaque écran (interface) est hérité de la classe mère Activity. Les activités utilisent les vues (Views) pour former les interfaces utilisateur graphiques qui afficheront l'information et répondront aux actions. Une application peut inclure plusieurs activités à la fois, ces activités peuvent ne pas avoir interface utilisateur.

2.1 Cycle de vie d'une activité

Dans le système Android, les activités sont gérées en tant qu'une pile d'activités. Lorsqu'une nouvelle activité est lancée, elle devient l'activité en cours, l'activité précédente reste toujours en arrière plan et ne retourne pas au premier plan tant que la nouvelle activité n'est pas terminée.

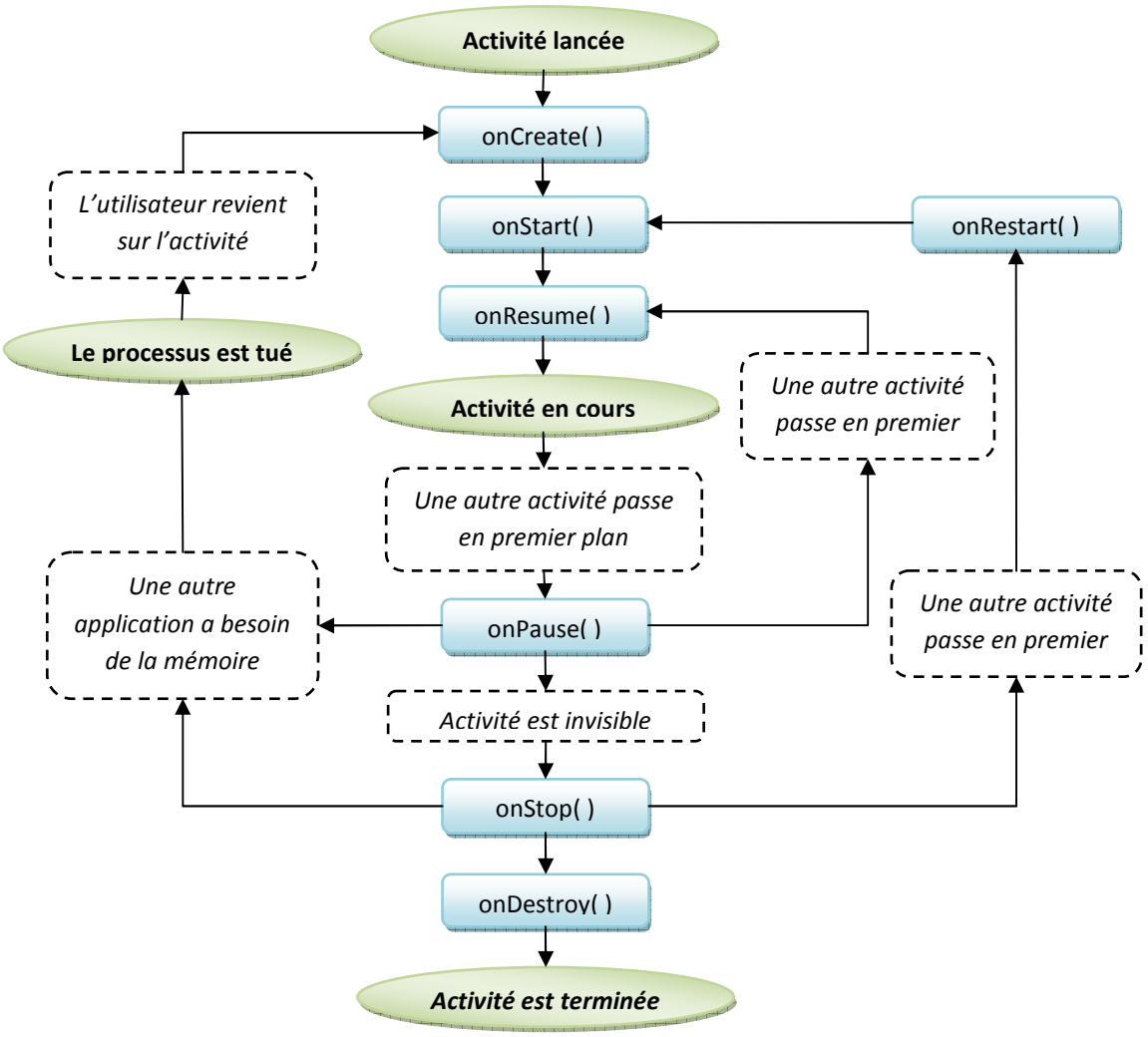


Figure6. Cycle de vie d'une activité

Une activité a principalement quatre états:

1- Une activité est active ou en cours d'exécution si elle est au premier plan de l'écran.
2- Une activité est mise en pause si elle n'est plus en premier plan mais reste visible (la nouvelle activité n'est pas affichée en plain écran ou elle est transparente). Par exemple, quand un utilisateur reçoit un appel téléphonique pendant qu'il regarde un vidéo. Le processus de lecture de vidéo est mis en pause (mais il reste visible). Une activité en pause est entièrement vivante, mais elle peut être détruite si le système aura besoin de plus de mémoire.
3- Une activité est arrêtée si elle est entièrement cachée (masquée) par une autre activité. Elle garde toutes les informations d'état, mais elle peut être détruite par le système s'il aura besoin de la mémoire.
4- Quand une activité est mise en pause ou arrêtée, elle doit être redémarrée et restaurée à son état précédent afin d'être à nouveau affichée à l'utilisateur.

En général, le passage entre les états de cycle de vie d'une activité est assuré par les méthodes de callback suivantes:

- **onCreate() / onDestroy()**: ces méthodes permettent la gestion des opérations à effectuer avant l'affichage de l'activité, et pendant la destruction de l'activité de la mémoire. La méthode **onCreate()** est invoquée lorsque l'activité est créée. Elle permet d'associer une vue à l'activité, initialiser ou récupérer les données persistantes.
- **onStart() / onStop()**: ces méthodes sont invoquées lorsque l'activité devient visible/invisible pour l'utilisateur.
- **onPause() / onResume()**: une activité peut être mise en pause (mais reste visible) lorsqu'une autre activité est en train de démarrer.
- **onRestart()**: cette méthode est appelée quand l'activité qui est passée par **onStrop()** est relancée. Par la suite, la méthode **onStart()** est appelée afin de différencier le premier lancement de re-lancement de la même activité.

2.2 Méthodes de la classe Activity

import android.app.Activity

```
public class Main extends Activity {

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.acceuil);
   }

 protected void onDestroy() {
       super.onDestroy(); }

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

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

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

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

Figure7. La classe Main.java

2.3 Méthodes de cycle de vie

Les méthodes onXxx sont invoquées au cours du cycle de vie d'une activité. Chaque appel de méthode correspond à un passage d'état dans le cycle de vie.

- onCreate(Bundle) : cette méthode est appelée à la création de l'activité. Le paramètre permet de récupérer l'état sauvegardé lors de l'arrêt de l'activité (si on fait une sauvegarde).
  Bundle : est une classe qui permet de sauvegarder des données (String, int, boolean,..) est de les faire passer comme paramètres entre les activités.
- onPause() : appelée quand l'activité n'est plus en premier plan.
- onDestroy() : invoquée quand l'activité se termine.

- onStart() : appelée quand l'activité démarre.
- onRestart() : appelée quand l'activité redémarre.
- onStop() : invoquée quand l'activité n'est plus visible.
- onResume() : appelée quand l'activité passe en premier plan.
- finish() : permet de terminer une activité

2.4 Méthodes relatives à l'interface

- setContentView(int id) :permet de charger une interface a partir d'un fichier XML, le paramètre définit ce fichier a travers son identifiant.
- findViewById(int id) : permet la récupération d'un élément d'interface (retourne un objet de classe View), le paramètre définit cet élément par son identifiant.
- showDialog(int id) : permet l'ouverture d'une fenêtre de dialogue. Le paramètre définit la fenêtre de dialogue par son identifiant

3 Le fichier Manifeste

Le fichier XML de configuration nommé AndroidManifest.xml est un composant indispensable pour une application Android. Ce fichier possède toutes les données nécessaires sur l'application pour que le système puisse la lancer. Ce fichier définit le nom du package de l'application, décrit les composants de l'application, définit les permissions nécessaires dont l'application a besoin et les permissions avec lesquelles les autres applications doivent avoir afin d'interagir avec cette application, déclare la version Android minimale supportée par l'application et les bibliothèques dont elle a besoin pour s'exécuter.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
      <uses-sdk />
      <uses-permission />
      <application>
            <activity>
                  <intent-filter>
                  <action />
                  <category />
                  <data />
                  </intent-filter>
            </activity>
            <service>
                  <intent-filter> . . . </intent-filter>
            </service>
            <receiver>
                  <intent-filter> . . . </intent-filter>
            </receiver>
            <provider>
                  <grant-uri-permission />
            </provider>
            <uses-library />
      </application>
</manifest>
```

3.1 Activity

Toutes les activités qui constituent l'application doivent être décrites par une balise <activity>. Les informations basiques de l'activité principale est de la forme suivantes:

```
<activity android:name=".Nom_de_Activite"
            android:label="Designation_de_Activite"
            android:icon="@drawable/Nom_du_fichier_icone"   >
       <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
       </intent-filter>
  </activity>
```

Les paramètres généralement utilisés dans la balise `<activity>` sont :

- ***name*** : définit le nom de la classe de l'activité.
- ***label*** : désigne le nom sous lequel elle va apparaître sur le terminal mobile.
- ***icon*** : indique l'icône à afficher sur le terminal mobile.

Les activités secondaires (lancées par l'activité principale) ne possèdent pas les propriétés *android:icon* et *android:label* car ces propriétés sont réservées uniquement à l'activité principale.

La balise *<intent-filter>* (ou filtre d'intention) permet de définir le type d'intention de l'activité. Elle offre au système la possibilité de savoir les *intents* à utiliser pour gérer les activités.

La forme générale de la balise <intent-filter> est la suivante:

```
<intent-filter>
      <action android:name="nom_d_action_1" />
       …
      <action android:name="nom_d_action_N" />

      <category android:name="nom_de_categorie_1" />
      …
      <category android:name="nom_de_categorie_N" />

      <data android:mimeType="nom_de_type_mime"
      android:scheme="protocole://hote:port/chemin" />

</intent-filter>
```

Un filtre d'intention peut contenir les rubriques suivantes :

- ***Action*** : permet de définir le type d'action réalisée par l'activité (affichage, édition …)
- ***Category*** : permet de préciser la catégorie d'action (par exemple CATEGORY_BROWSABLE indique une activité qui peut être appelée par un navigateur)
- ***Data*** : permet de définir le type de données envoyées et reçus par l'activité ainsi que le protocole (http, content, file …) utilisé.

Android propose des valeurs prédéfinies pour ces diverses rubriques:

− *Actions*

android.intent.action.MAIN activité principale d'une application

android.intent.action.CALL appel téléphonique

android.intent.action.WEB_SEARCH recherche sur le WEB

− *Catégories*

android.intent.category.LAUNCHER activité proposée au lancement par Android

android.intent.category.BROWSABLE pour afficher une information référencée par un lien.

3.2 Service

Un service est un composant d'application Android qui s'exécute en arrière-plan pour réaliser des taches continues ou qui n'a pas besoin de l'intervention de l'utilisateur comme les recherches sur le réseau ou les traitements qui doivent encore s'exécuter même lorsque les activités de l'application ne sont pas actives ou visibles. Un service ne fournit pas d'interface utilisateur. Par exemple, un service peut lire un fichier audio en arrière-plan alors que l'utilisateur se trouve dans une application différente, ou il peut extraire des données sur le réseau sans bloquer l'interaction de l'utilisateur avec une activité.

3.3 Fournisseur de contenu

Propose un niveau d'abstraction pour toutes les données stockées sur le dispositif mobile et accessibles aux différentes applications. Il permet de gérer et sauvegarder les données de l'application. Il offre aux autres applications la possibilité d'interroger ou même de modifier les données (s'il le permet). L'utilisateur peut configurer son propre fournisseur de contenu pour autoriser l'accès à d'autres applications. Les fournisseurs de contenu sont également utiles pour lire et écrire des données privées à votre application et qui ne sont pas partageables. Les dispositifs mobiles Android ont plusieurs fournisseurs de contenus tels que les médias et les contacts.

3.4 Intent

Intent est un objet qui permet de diffuser des messages en demandant la réalisation d'une action. Les messages de Intent contenant des données émis par Android (l'OS, une autre application, un autre service) pour prévenir les applications en cours d'exécution de la survenue d'un événement comme la réception d'un SMS, ou un déplacement du GPS. L'objet Intent permet aux applications de fournir ou demander des services ou des données. La communication est faite à travers tout le système et peut viser précisément une activité ou un service. En générale, un objet Intent offre trois utilisations: le lancement d'une activité au sein de l'application courante, la demande d'autres applications et l'envoie des données.

Voici un exemple simple d'une intention implicite qui ouvre une URL de page Web dans le navigateur:

```
String url = "http://univ-mosta.dz";
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse(url));
startActivity(intent);
```

3.5 Filtre intent

Un ou plusieurs filtres d'intentions peuvent être associés à chaque activité ou service afin de permettre à Android de choisir une activité ou service en mode implicite. Un Intent Filter définit les types d'intention qui peuvent être assurés par une activité, ou un service. Il déclare donc les capacités d'un composant. Les composants Android stockent leurs Intent Filter selon deux manières : soit statiquement dans le fichier AndroidManifest.xml, soit dynamiquement dans le code (comme dans le cas d'un récepteur de diffusion). Un filtre d'intentions est décrit par : une catégorie, une action et des filtres de données. Il peut aussi avoir des métadonnées supplémentaires.

3.6 Permission

Pour qu'une application puisse utiliser certaines ressources, elle doit définir les autorisations (permissions) spécifiques aux ressources demandées à travers la balise <uses-permission>.

Parmi les principales permissions on peut citer:

Accès aux données personnelles

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

Téléphonie

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

Réseau

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

Access au matériel

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

4 Ressources

4.1 Classe R (Classe de référencement des ressources)

Cette classe permet de référencer les ressources pour l'utiliser dans le code des activités de l'application. L'outil ***aapt*** permet de générer automatiquement un identifiant pour chaque ressource (image, fichier, variable, menu, etc.) dans la classe *R.java.* Les ressources de l'application sont stockées dans répertoire **res**.

```
package com.example.test;

public final class R {

    public static final class drawable {
        public static final int ic_launcher=0x7f020002;      }
    public static final class id {
        public static final int add=0x7f090008;              }
    public static final class layout {
        public static final int activity_main=0x7f030000;    }
    public static final class menu {
        public static final int main=0x7f080000;             }
    public static final class string {
        public static final int action_settings=0x7f060002;  }
    }
```

Les ressources sont désignées par leur identifiant dans la classe **R** sous la forme :

`R.type.nom`.

Par exemple, une image "ma_photo.png" placée dans drawable-hdpi sera désignée par R.drawable.ma_photo

4.2 Type de ressources

Plusieurs types de données des ressources (entiers, booléens, chaînes de caractères, etc. et des tableaux) peuvent être définis dans des fichiers xml comme la forme suivante:

```
<?xml version="1.0 " encoding= "utf-8 "?>
      <resources>
            <color name= "couleurfond">#AA7B03</color>
            <integer name= "age">20</integer>
            <integer-array name= "codes_postaux">
                  <item>27000</item>
                  ...
                  <item>31000</item>
            </integer-array>
            <string name= "titre">Un titre</string>
            <string-array name= "villes">
                  <item>Mosta</item>
                  ...
                  <item>Alger</item>
            </string-array>
            <bool name="actif">true</bool>
            <dimen name "taille">55px</dimen>
      </resources>
```

4.3 Référencer une ressource dans un fichier XML

Une ressource est référencée dans un fichier XML par :

"@[package:]type/identificateur"

Par exemple : @string/*titre* fait référence à une chaîne décrite dans un fichier XML placé dans le répertoire res/values (le nom de ce fichier n'a aucune importance) et définie par :

<string name="*titre*">contenu de cette chaine</string>

4.4 Récupérer une ressource dans le code

Dans le code java, les ressources sont désignées par leur identifiant dans la classe. Pour accéder au contenu de ressources on fait appel à la classe Resources.

import android.content.res.Resources

L'instance de la classe ***Resources*** est obtenue par la méthode *getResources()* de l'activité. L'accès aux ressources ce fait a travers les méthodes de cet objet qui acceptent en paramètre un identifiant de la forme *R.type.nom*, ces méthodes sont :

- boolean getBoolean(int) retourne le booléen désigné
- int getInteger(int) retourne l'entier désigné
- String getString(int) retourne la chaîne désignée

Par exemple : String monTitre = getResources().getString(R.string.*monTitre*) retourne la valeur de la chaîne de caractère définie dans un fichier XML par :

<string name="*monTitre*">…..</string>

5 Conclusion

Dans ce chapitre on a présenté d'une manière générale les différents composants d'une application mobile tels que les activités, ressources, et le fichier manifest ainsi que la manière d'interaction entre ces composants.

Questions

- Quelle est la différence entre un ***démarrage implicite*** et un ***démarrage explicite*** d'une activité ?
- Donner la syntaxe de référencement d'une ressource de type image (monImage.png) dans un fichier d'interface layout.xml

Chapitre IV :

Les interfaces graphiques Android

1 Introduction

La création d'une interface en Android se fait par la création de deux éléments principaux:

- Une définition de l'interface utilisateur (widget[13], conteneurs, etc.) dans un fichier XML ;
- Une définition du logique métier (comportement de l'interface) dans une classe d'activité.

Cela permet d'assurer une séparation entre la présentation (l'interface) et la logique fonctionnelle de l'application.

1.1 Propriétés générales de positionnement XML

Un positionnement XML est une spécification des relations existant entre les composants graphiques (widgets) et leurs conteneurs (layouts) exprimée sous la forme d'un document XML. Chaque fichier XML contient une arborescence d'éléments définissant le layout des widgets et les conteneurs qui composent la vue (view). Les attributs de ces éléments indiquent l'apparence d'un widget ou le comportement d'un layout.

Tous les éléments de l'interface (vues et conteneurs) doivent définir au moins les deux attributs suivants :

- ***android:layout_width*** : permet de définir la largeur de la vue.
- ***android:layout_height*** : permet de définir la hauteur de la vue.

Ces attributs peuvent avoir les valeurs suivantes:

- ***"wrap_content"*** : L'élément ne prend que la place qui lui est nécessaire en largeur et hauteur pour s'adapter à la taille de son contenu.
- ***"match_parent"*** : spécifie que l'élément (vue ou layout) doit prendre toute la place disponible sur la largeur et la hauteur. L'élément prend la taille du son père.
- ***"valeur*dp"** : une taille fixe, ex : "100dp" mais c'est peu recommandé. Le dp (*dots per inch*) est une unité de taille indépendante de l'écran. Par exemple, 100dp font 100 pixels sur un écran de 100 dpi (100 *dots per inch*).

1.2 Identifiant

Un identifiant peut être attribué à chaque élément décrit dans un fichier XML afin de le référencé a partir du code. Cet attribut peut avoir les deux notations :

- ***@+id/nomID*** pour créer l'identifiant *nomID* à une vue.
- ***@id/nomID*** pour référencer un identifiant déjà défini dans le fichier R.java

Par exemple, android:id="@+id/nomID" permettra de retrouver cet élément par findViewById(R.id.nomID).

[13] Les composants graphiques sont nommés *widget* (contraction des mots anglais *window* et *gadget*)

2 Layout (conteneur)

Dans la littérature, un conteneur, un gabarit, ou encore mise en page, est une extension de la classe mère *ViewGroup*. Un conteneur permet de positionner les vues et/ou layouts dans une interface Android. On peut imbriquer des layouts les uns dans les autres, cela permet de créer des interfaces évoluées. Tous les éléments contenus dans le layout sont appelés des enfants. Par exemple, un layout enfant est inclut dans le layout parent (il faut noter que l'écran est le parent du layout parent). Les vues et les layouts sont disposés selon le type de layout choisi :

- **LinearLayout** : permet d'aligner les éléments (vues, layouts) de gauche à droite ou de haut en bas. L'attribut *orientation* permet de préciser dans quel sens s'affichent les éléments : avec la valeur *horizontal*, l'affichage est de gauche à droite alors qu'avec la valeur *vertical* affichage est de haut en bas.
- **RelativeLayout** : permet de positionner les éléments les uns par rapport aux autres. Le premier élément sert de référence pour les autres.
- **TableLayout** : permet de positionner les vues en lignes et colonnes comme un tableau.
- **AbsoluteLayout** : ce layout permet de définir les coordonnées exactes des éléments qui le composent.

L'extrait de code suivant montre comment inclure un LinearLayout dans un fichier layout XML. Il faut noter que le fichier XML qui contient l'interface est stocké dans le répertoire *res/layout.*

```
<LinearLayout   xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
   <!— Ajoutez ici d'autres vues ou layouts. Elles sont considérées
      comme des "enfants" de LinearLayout -->
 </LinearLayout>
```

L'attribut *android:orientation* permet de spécifier si les vues enfant doivent être affichées en ligne ou en colonne.

Remarque : Plusieurs éléments et d'attributs d'Android sont normalisés. Afin de les distinguer, ils sont regroupés dans l'espace de nom (namespace) android. Dans la norme XML, l'espace de nom n'est pas appliqué aux attributs par défaut, donc on doit ajouter le préfixe suivant sur chaque attribut.

L'élément racine doit déclarer l'espace du nom XML d'android.

```
xmlns:android=http://schemas.android.com/apk/res/android
```

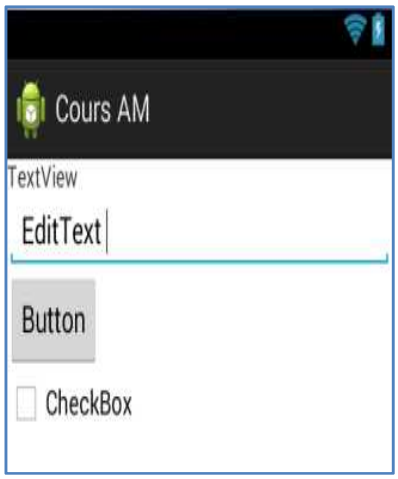
Tous les autres éléments sont des enfants de la racine (parent) et héritent cette déclaration d'espace du nom.

3 Vue (view)

C'est la classe mère de tous les composants graphiques d'Android qui regroupe l'affichage ainsi que la gestion des évènements. Toutes les vues se trouvent dans le package *android.widget*. Ces vues évoluent avec les versions d'Android, certaines changent, et d'autres disparaissent.

Le tableau suivant résume quelques vues basiques :

```
<TextView    android:id= "@+id/text"
             android:layout_width= "wrap_content"
             android:layout_height= "match_parent"
             android:text= "@string/letexte" />

<EditText    android:id= "@+id/edtext"
             android:layout_width= "wrap_content"
             android:layout_height= "match_parent"
             android:inputType= "textMultiLine"
             android:hint= "@string/letexte" />

<Button      android:id= "@+id/b1"
            android:layout_width= "wrap_content"
            android:layout_height= "wrap_content"
            android:text= "@string/boutonTexte" />

<CheckBox    android:id= "@+id/c1"
            android:layout_width= "wrap_content"
            android:layout_height= "wrap_content"
            android:text= "@string/texte"
            android:checked="true" />
```



4 Associer une interface à une activité (logique métier)

Chaque interface est gérée par une instance d'une sous-classe d'Activity. Le contenu de l'interface est chargé lorsque l'activité est instanciée. La méthode *onCreate* est redéfinie (surchargée) pour spécifier la définition de l'interface à afficher via la méthode *setContentView*.

```
import android.app.Activity;
import android.os.Bundle;

public class Main extends Activity {
     @Override
     public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
       }
}
```

La méthode *setContentView* prend en paramètre l'identifiant de l'interface (la vue) à afficher dans l'écran : R.layout.main, c'est un entier identifiant un layout.

Pour accéder à un composant graphique depuis le code et pouvoir le manipuler, on doit d'abord récupérer une instance de l'objet avec la méthode *findViewById*(). Une fois l'objet récupéré, on peut le manipuler de la même façon que n'importe quel objet.

Le code suivant récupère le composant graphique *TextView* nommé *tv* dans l'interface, puis utilise la méthode *setText* pour changer le contenu de son texte.

```
TextView tv = (TextView) findViewById(R.id.text);
tv.setText("Mon texte");
```

5 Gestion des évènements sur les widgets

Toute action faites par l'utilisateur pour interagir avec l'interface graphique (le click, le maintien du click, les touches, etc.) est perçues comme un événement. Cet événement est intercepté par un mécanisme basé sur la notion d'écouteur (***listener***).

Un *listener* permet d'attribuer un événement à une méthode pour l'appeler dés que cet événement sera produit.

Par exemple, la classe *android.view.View.OnClickListener* permet d'associer un écouteur d'événements de type click, et la méthode à surcharger pour traiter ces événements est *onClick(View).*

La méthode *setOnClickListener(OnClickListener)* associée un écouteur d'événements aux actions de click sur une vue.

Il existe plusieurs manières pour définir un écouteur (listener) dans une activité:

Écouteur privé anonyme : c'est une classe qui est définie lorsque la méthode setOnClickListener est appelée. Elle ne peut contenir qu'une seule méthode.

```
Button btn = (Button) findViewById(R.id.btn1);
btn.setOnClickListener(new View.OnClickListener() {
     public void onClick(View btn) {
       // code lancé lorsque le bouton est cliqué
     }
   });
```

Écouteur privé : c'est une classe privée définie pour implémenter l'interface OnClickListener dans l'activité. Cette classe est instanciée pour fournir un nouvel écouteur.

```
private class BtnOK implements OnClickListener {
   public void onClick(View btn) {
     // code lancé lorsque le bouton est cliqué
     }
};
public void onCreate(...) {
 ...
   Button btn=(Button)findViewById(R.id.btn1);
   btn.setOnClickListener(new BtnOK());
}
```

L'activité elle-même en tant qu'écouteur : consiste à définir ***this*** comme écouteur et d'indiquer qu'elle implémente l'interface OnClickListener.

```
public class Main extends Activity implements OnClickListener {
 public void onCreate(...) {
      ...
    Button btn=(Button)findViewById(R.id.btn1);
    btn.setOnClickListener(this);
  }
 public void onClick(View btn) {
   //code lancé lorsque le bouton est cliqué
  }
}
```

Différenciation des émetteurs : lorsque le même écouteur est utilisé pour plusieurs vues pour réaliser des actions différentes, on les différencies par leur identifiant obtenu la méthode ***getId()*** .

```
public void onClick(View v) {
  switch (v.getId()) {
     case R.id.btn1:
       //code lancé lorsque le bouton est cliqué
     break;
     case R.id.btn2:
       //code lancé lorsque le bouton est cliqué
     break;
    }
}
```

Écouteur dans le layout : On peut ajouter dans le fichier de positionnement xml, la méthode qui sera lancée sous une certaine action sur un composant graphique.
L'attribut **android:onClick** d'un composant graphique définit le nom de la méthode qui sera appelée lorsqu'on clique sur ce composant.
Par exemple, on ajoute l'attribue suivant dans le fichier de positionnement xml:

```
<Button  android:id="@+id/btn1"
            ...
         android:onClick="BtnOK">
 </Button>
```

Dans l'activité qui charge le fichier de positionnement qui contient le Button, on peut écrire la méthode suivante:

```
public void BtnOK (View v){
   //code lancé lorsque le bouton est cliqué
          }
```

D'autres écouteurs d'événements peuvent être gérées par les classes de base des interfaces:

***OnLongClickListener* :** permet de définit un écouteur d'événement de type click long. Pour traiter cet événement, il suffit de surcharger la méthode: **onLongClick(View)**

***OnKeyListener**:* définit un écouteur d'événement de type clavier. La méthode à surcharger pout traiter cet événement est : **onKey(View, int, KeyEvent)**

Le deuxième paramètre représente le code de la touche tapée, et le troisième est l'événement clavier. Le code de la touche tapée peut prendre les valeurs suivantes :

- KEYCODE_0 à KEYCODE_9 pour les chiffres
- KEYCODE_A à KEYCODE_Z pour les majuscules

KeyEvent : associée aux événements clavier. Parmi ces méthodes :

- getAction () : retourne un entier qui peut avoir les valeurs ACTION_DOWN ou ACTION_UP indiquant l'action faite sur la touche.
- getRepeatCount() : retourne le nombre de répétitions lorsque la touche est maintenue.
- getKeyCode() : renvoie le code de la touche.

OnTouchListener : définit un écouteur d'événement de type touché. Le surcharge de la méthode **onTouch(View, MotionEvent)** permet de traiter cet événement.

Le second paramètre est l'événement de touché.

MotionEvent : attribuée aux événements de déplacement (souris, écran tactile). Ses méthodes sont les suivantes :

- getAction() : retourne un entier qui peut valoir les valeurs suivantes : ACTION_DOWN , ACTION_UP, ACTION_MOVE ou ACTION_OUTSIDE
- getPressure() retourne un valeur réel entre 0 et 1 indiquant la force de pression appliquée sur l'écran par le touché de l'utilisateur.
- getX() et getY() : retournent un réel entre -1 et 1 précisant les coordonnées (resp en x et en y).

6 Conclusion

Le succès d'une application mobile repose sur une interface ergonomique et intuitive. La plate-forme Android a rendu la création d'interface simple et efficace. Dans ce chapitre nous avons présenté les éléments de base qui permettent de concevoir une interface graphique d'une application mobile.

Questions

- Compléter le code xml (interface) suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  android:layout_width = …
  android:layout_height = …
  android: orientation = …  >
        <CheckBox   android:id = …
                    android:layout_width = …
                    android:layout_height = …
                    android:text = …  />
</LinearLayout>
```

Chapitre V :

Interactions entre les composants Android

1 Introduction

L'intention d'une action dans Android est exprimée par les échanges des messages internes à travers l'objet ***Intent*** qui est une description abstraite d'une opération à réaliser. Intent permet de : (i) lancer une activité dans l'application courante, (ii) solliciter d'autres applications et (iii) d'envoyer des informations.

Les principaux paramètres d'un Intent pour la réalisation de l'action sont les suivants:

- La description de l'action à effectuer et les données avec lesquelles agir.
- La catégorie du composant cible et les instructions d'exécution de l'action.

D'autres paramètres, optionnelles, peuvent être utilisées selon les besoins :

- *Le type* : pour spécifier quel est le type des données incluses (imposer un type particulier).
- *Les extras* : pour ajouter des données à l'intent afin de les faire passer entre les composants.
- *Les flags* : permettent la modification du comportement de l'intent.

Il existe deux principaux types d'Intents :

- **Explicite** : indiqué le composant qui fournit la classe exacte à exécuter.
- **Implicite** : Ne définie pas le composant mais fournit les informations nécessaires qui permettent au système de déterminer les composants correspondant à cette action.

2 Lancer une activité de manière explicite

Pour lancer une activité sans attendre de retour, on utilise la méthode ***startActivity*** qui prend comme paramètre une instance de la classe ***Intent*** désignant le type de classe de l'activité à exécuter.

```
Intent monIntent = new Intent(getApplicationContext(),Activite_A_Lancer.class);

startActivity(monIntent);
```

La classe Intent prend les paramètres suivants :

- **Context PackageContext** : le contexte à partir duquel l'Intent est créé et sera envoyé. Il fait référence à l'activité en cours. On peut l'avoir on utilisant la fonction *getApplicationContext()* ou bien le mot clé *this.*
- **Class<?> cls** : un type de classe Java héritant de la classe Activity, il précise le nom de la class à lancé, dans notre cas c'est Activite_A_Lancer.class.

NB. Il ne faut pas oublier la déclaration de l'activité a lancée dans le fichier de configuration *AndroidManifest.xml* de l'application.

```
<application ...>
<activity android:name=".Activite_A_Lancer" />
</application>
```

Sinon, une exception du type *ActivityNotFoundException* sera déclenchée lorsque la méthode startActivity est appelée.

La méthode ***startActivityForResult()*** permet de lancer une activité et de retourner des résultats de cette dernière.

Le lancement est de la forme suivante :

```
private static final int CODE_DE_LA_REQUETE = 1;
        ...
Intent monIntent = new Intent(getApplicationContext(), ActivityEnfant.class);
startActivityForResult(monIntent, CODE_DE_LA_REQUETE);
```

La constante CODE_DE_LA_REQUETE (*requestCode*) permet d'identifier quelle activité a provoqué le retour de valeur. Pour traiter plusieurs valeurs de retour dans la même méthode, on peut utilisé l'instruction switch(requestCode). Si la valeur est inférieure à 0, alors la méthode ne retourne rien.

Pour renvoyer la valeur de retour à l'activité principale, l'activité enfant (secondaire) appel la méthode **setResult** qui prend en paramètre le code de retour. Android propose plusieurs valeurs par défaut telles que **RESULT_OK** et **RESULT_CANCELED**. Par exemple, dans une méthode onClick() de l'activité enfant (`ActivityEnfant`) on fait le code suivant : **setResult**(RESULT_OK);

Pour récupérer la valeur de retour envoyée par une activité enfant (`ActivityEnfant`), on utilise la méthode ***onActivityResult*** de l'activité parent qui a appelée la méthode *startActivityForResult.*

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch(requestCode){
      case CODE_DE_LA_REQUETE:
         switch(resultCode){
           case RESULT_OK:
             // faire quelque chose
           return;
           case RESULT_CANCELED:
            // faire quelque chose;
           return;
          }
 }
```

La méthode ***onActivityResult*** utilise trois paramètres pour traiter les valeurs de retour :

- ***int requestCode*** : c'est l'identifiant de l'activité qui a appelée la méthode (par exemple, CODE_DE_LA_REQUETE). Il est utilisé comme paramètre dans la méthode

startActivityForResult lors de lancement de l'activité enfant.

- ***int resultCode :*** représente la valeur de retour envoyée par l'activité enfant (ActivityEnfant) pour indiquer son état à la fin de la transaction. C'est une valeur constante. Elle est soit prédéfinie dans la classe Activity (RESULT_OK, RESULT_CANCELED, etc.), soit définit par le développeur.
- ***Intent data*** : c'est un objet qui permet de communiquer des données entre les activités.

3 Lancer une activité de manière implicite

Intent implicite ne définit pas le composant cible, mais possède assez d'informations pour aider le système à déterminer le composant le mieux approprié pour réaliser cette intention. Les composants récepteurs sont fournis par Android, ou par d'autres applications téléchargées sur le Play Store. Les données de cet Intent sont formatées par un URI[14].

Par exemple, pour passer un appel téléphonique, les paramètres de l'intent sont composés d'une action native et d'un URI comportant le numéro à appeler.

```
Uri uri = Uri.parse("tel:0451234xx");
Intent monIntent = new Intent(Intent.ACTION_DIAL, uri);
startActivity(monIntent) ;
```

Il faut ajouter la permission *android.permission.CALL_PHONE* dans le fichier AndroidManifest.xml afin de permettre l'exécution de cet intent.

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

Le système Android propose plusieurs actions natives comme :

- ACTION_WEB_SEARCH : permet d'effectuer une recherche sur Internet avec l'URI indiquée en paramètre comme requête.
- ACTION_VIEW : permet de lancer une action pour visualiser l'élément identifié par l'URI spécifiée en paramètre. Si l'adresse commence par **http**: le navigateur web est lancé, et si elle commence par **tel**: c'est l'interface de composition de numéro qui sera lancé, sinon si l'adresse début par **geo**: c'est Google Map qui se lancera.
- ACTION_DIAL : pour afficher l'interface de composition des numéros de téléphone. Cette interface peut être pré-remplie par les informations contenues dans l'URI indiquée en tant paramètre.

[14] Un URI (*Uniform ressource Identifier*) est un identifiant unique qui permet d'identifier une ressource de façon non ambigüe sur un réseau.

4 Passage de paramètres

Pour transmettre des données entre les activités, l'intent utilise les méthodes **putExtra** et **getExtras.**

L'envoie de données se fait par la méthode **putExtra** et l'extraction à travers la méthode **getExtras.**

```
public Intent putExtra("CLE", VALEUR) ;
```

L'échange des données se fait par l'attribution d'une clé unique (une chaîne de caractères) à une valeur à échanger. La méthode putExtra support tous les types de données primitifs: int, float, String, etc.

```
Intent monIntent = new Intent(getApplicationContext(), monActivity.class);
       monIntent.putExtra("name","Ali");
       monIntent.putExtra("age",24);
       startActivity(monIntent);
```

Pour récupérer l'Intent qui véhicule les données envoyées, on utilise la méthode **getIntent().** Les données sont encapsulées dans un objet de type **Bundle**. Les données de Bundle sont récupérées par la méthode **getExtras**. Par la suite, un extra est associé à une entrée par la méthode:

```
getTypeInput("CLE", VALEUR_Par_DEFAUT); ou getTypeInput("CLE");
```

VALEUR_Par_DEFAUT est la valeur retournée s'il n'existe pas d'extra associé à CLE dans l'Intent.

get*TypeInput* correspondant au type de la donnée par exemple **getString** pour une donnée de type String, **getInt** pour une donnée de type Int, etc.

```
protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
         Bundle extra = this.getIntent().getExtras();
        if(extra != null){
         String nom = extra.getString("name");
            Int age = extra.getInt("age",0);
       }
 }
```

5 Conclusion

L'intention d'une action dans Android s'exprime par l'envoi et la réception de messages d'une Intention entre les composants de la même application (activité, service, etc.) ou ceux de toute autre application. Nous avons vue dans ce chapitre que l'Intent permet de véhicule toutes les informations nécessaires à la réalisation d'une action.

Chapitre VI :

Interfaces graphique avancées

1 Menu

Un menu est une liste d'éléments (items) qui permet de proposer des fonctionnalités supplémentaires qui n'apparaissent pas par défaut à l'écran, et de mieux gérer la taille limitée de l'écran de dispositif mobile. Chaque menu est associé à une seule activité. La sélection d'un élément (item) dans le menu déclenche une *callback*[15].

Pour réaliser un menu, il faut définir :

- Un fichier de définition d'interface res/menu/*nom_du_menu*.xml,
- Les méthodes de *callbacks* pour gérer les menus : création et réaction.

1.1 La description XML de Menu

L'élément <item> prend en charge plusieurs attributs qu'on peut utiliser pour définir l'apparence et le comportement d'un élément. Les éléments du menu (res/menu/*nom_du_menu*.xml) suivant incluent les attributs nécessaires:

```
<menu xmlns:android="..." >
      <item android:id="@+id/monMenu"
            android:icon="@drawable/icon_monMenu"
            android:showAsAction="ifRoom"
            android:title="@string/monMenu"/>
      <item android:id="@+id/autre_Menu" ... />
      ...
</menu>
```

Il faut définir les attributs ; identifiant (id), l'icône (icon) et le titre de menu (title). L'attribut ***showAsAction*** spécifie quand et comment cet item doit apparaître en tant qu'élément d'action dans la barre d'application. Il prend les valeurs suivantes :

- **ifRoom :** Ne placer cet élément dans la barre de l'application que s'il y a de la place.
- **always** : Placer toujours l'élément dans la barre d'applications (Si l'appariation de l'élément dans la barre d'actions est indispensable)**.**
- **never :** Ne maitre jamais cet élément dans la barre d'applications.

1.2 Les méthodes de callback

Deux méthodes sont utilisées. La première pour afficher le menu, la seconde pour réagir quand l'utilisateur sélectionne un item de menu.

[15] Une fonction de rappel (*callback* en anglais) est une fonction passée dans une autre fonction en tant qu'argument, qui est ensuite invoquée à l'intérieur de la fonction externe pour accomplir une action.

Dans l'activité, afin d'utiliser *nom_du_menu*.xml en tant qu'icones dans ActionBar, il suffit de l'indiquer à *MenuInflater* dans la méthode **onCreateOptionsMenu** afin d'ajouter les items au menu.

Un MenuInflater est un lecteur/traducteur de fichier de description XML. La méthode ***inflate*** permet de crée les vues.

```
public boolean onCreateOptionsMenu(Menu menu){

        getMenuInflater().inflate(R.menu.nom_du_menu, menu);
```

Pour récupérer le click de l'utilisateur sur une action, il suffit de surcharger la méthode d'activité **onOptionsItemSelected(MenuItem item).** En effectuant un *switch* sur l'identifiant de l'item, on peut savoir lequel a été sélectionné.

```
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
       case R.id.monMenu:
         // faire quelque chose
       return true;
       case R.id.auterMenu:
         // faire quelque chose
       return true;
       ...
       default:
       return super.onOptionsItemSelected(item);
                          }
}
```

2 Menu contextuel

Un menu contextuel propose des actions qui affectent un élément ou un cadre de contexte spécifique dans l'interface utilisateur. On peut attribuer un menu contextuel à n'importe quelle vue, mais il est le plus souvent utilisés pour les éléments d'une ImageView, ListView, GridView ou d'autres collections de vues ou l'utilisateur peut réaliser des actions directes sur chaque élément. La méthode **registerForContextMenu (View v)** *permet d'associer* un menu contextuel à un composant graphique (vue). Quand l'utilisateur fait un click long sur cette vue, ce menu contextuel s'ouvrira.

```
protected void onCreate(Bundle savedInstanceState){
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
   ImageView img = (ListView)findViewById(R.id.image);
   registerForContextMenu(img);
       ....
 }
```

On peut aussi faire **img.setOnCreateContextMenuListener(this)** ou lieu de la méthode registerForContextMenu(img);

La première méthode est utilisée pour l'affichage de menu contextuel, elle permet d'expanser (inflate) le menu contextuel qui est stocké dans : *res/menu/menu_context.*

```
public void onCreateContextMenu(ContextMenu monMenu, View v,
                                ContextMenuInfo menuInfo){
      super.onCreateContextMenu(menu, v, menuInfo);
      getMenuInflater().inflate(R.menu.menu_context, monMenu);
 }
```

monMenu : est le menu à construire, ***v*** : c'est la vue sur laquelle le menu a été appelé et ***menuInfo*** : indique sur quel élément d'un adaptateur (par exemple liste) a été appelé le menu.

La deuxième méthode de callback permet de répondre à l'utilisateur quand il choisit un item du menu.

```
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo)item.getMenuInfo();
  switch (item.getItemId()) {
      case R.id.ajouter:
         . . .
      return true;
      case R.id.modifier:
         . . .
      return true;
      }
      return false;
}
```

L'objet ***AdapterContextMenuInfo*** renvoie l'identifiant de ce qui est sélectionné (c-a-d ce qui a fait apparaître le menu contextuel).

3 Annonces

Afin d'annoncer a l'utilisateur des informations sans le déranger alors qu'il utilise une autre application ou qu'il rédige un SMS par exemple. Android propose le concept de toast, un message qui s'affiche en quelques secondes.

Toast : permet d'afficher un message de notification pendant quelques secondes de tel sort que l'utilisateur soit informé de la réalisation d'une action. Il utilise trois paramètres : *Context* de l'application, un *message textuel* (chaine de caractères) et une *durée* du temps (courte : 2 seconds et long : 5 seconds).

```
public static Toast makeText (Context context, CharSequence text, int duree);
```

Le toast est affiché par la méthode **show()**, comme dans l'exemple suivant :

```
Toast.makeText(getApplicationContext(),"Mon message",
                                    Toast.LENGTH_SHORT).show();
```

La durée d'affichage de message peut être prolongée avec Toast.LENGTH_LONG.

4 Dialogue

Une boîte de dialogue (Dialog) est une petite fenêtre qui ne remplit pas complètement la taille de l'écran et qui invite l'utilisateur à prendre une décision ou à entrer des informations supplémentaires avant de pouvoir continuer. Chaque boite de dialogue est composée de 3 parties :

- *Titre* : permet d'attribuer un titre à l'action de l'utilisateur.
- *Contenu* : peut inclure un simple texte ou un *layout* plus complexe comme un formulaire. Il définit ce que peut faire l'utilisateur.
- *Actions* : support 3 boutons (positive, négative, neutre) qui permettent soit de valider la décision de l'utilisateur, soit l'annuler ou de ne rien faire.

Il existe plusieurs types de dialogues, comme les dialogues d'alerte, et les dialogues généraux.

Un dialogue d'alerte est construit à l'aide de la classe **AlertDialog.Builder**. Elle permet de créer un *builder* qui à son tour crée la boite de dialogue.

```
AlertDialog.Builder dialog = new AlertDialog.Builder(this);
    dialog.setTitle("Confirmation");
    dialog.setMessage(" Voulez vous acceptez cette invitation?");
    dialog.setPositiveButton("Oui", new DialogInterface.OnClickListener(){
              public void onClick(DialogInterface dialog, int idBtn) {
                       // faire quelque chose
                                 }
                               });
    dialog.setNeutralButton("Plus tard", null);
    dialog.setNegativeButton("Non", null);
    AlertDialog confirm = dialog.create();
    confirm.show();
```

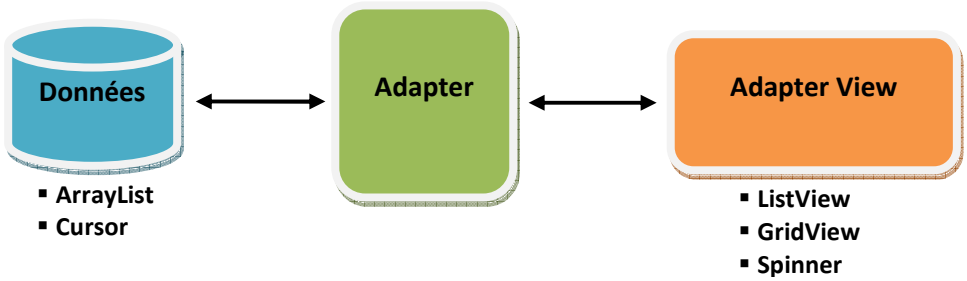
Pour demander des informations à l'utilisateur sans avoir besoin de créer une autre activité, on peut utiliser un ***dialogue personnalisé***. Pour ce faire, il faut définir un ***layout*** du dialogue contenant tous les textes, à part le titre, et au moins un bouton pour valider, mais on peut simplement fermer le dialogue avec le bouton back (retour) du smartphone. Ce layout est ajouté avec la méthode :

```
dialog.setContentView(R.layout.nom_layout);
```

5 ListView

Android ListView est une vue qui contient un groupe d'éléments et s'affiche dans une liste déroulante. ListView est implémenté en important la classe *android.widget.ListView*.

ListView utilise des classes d'adaptateurs qui ajoutent le contenu d'une source de données (un tableau de chaînes, un tableau, une base de données, etc.) à ListView. L'adaptateur relie les données entre un AdapterView et d'autres vues (ListView, ScrollView, etc.).



La gestion des listes est divisée en deux parties :

- **Adapter** : c'est un objet qui gère uniquement les données. Il joue le rôle d'intermédiaire entre les données et la vue qui représente ces données. Android propose trois types d'adaptateurs simples :
    1. *ArrayAdapter* : permet d'afficher les données simples (texte).

2. *SimpleAdapter* : affiche plusieurs informations pour chaque élément (nom et prénom par exemple) de la liste.
3. *CursorAdapter* : expose le contenu d'une base de données SQLite locale.

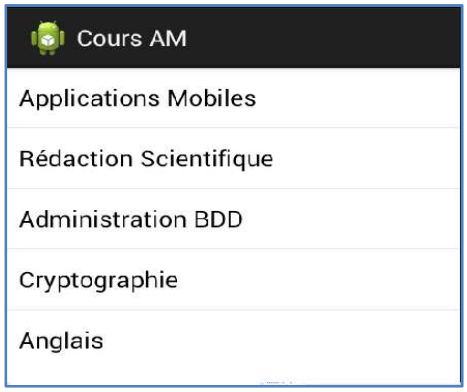
- **AdapterView :** gère l'affichage et l'interaction avec l'utilisateur, mais il ne permet pas d'effectuer d'opération de modification des données. Android propose trois types AdapterView :

1. *ListView* : affiche de manière simple des éléments les uns après les autres.
2. *GridView* : permet d'organiser les éléments sous la forme d'une grille.
3. *Spinner* : affiche les éléments dans une liste défilante.

Pour afficher une liste à partir d'un ensemble de données :

- On attribue à l'adaptateur une liste d'éléments à traiter.
- l'adaptateur crée une vue pour chaque élément en se basant sur les informations fournies avec le layout.
- Ensuite, les vues sont fournies à un AdapterView pour être affichées dans l'ordre fourni et avec le layout correspondant. L'AdapterView possède également un layout pour le personnaliser.

Pour utiliser un *ArrayAdapter* de base, il suffit d'initialiser l'adaptateur et de l'attacher à ListView.



ArrayAdapter requiert une déclaration du type de l'élément à convertir en vue (une chaîne dans ce cas), puis accepte trois arguments: contexte (instance d'activité), disposition de l'élément XML et tableau de données.

La mise en page choisie pour chacun des éléments est basée sur le layout ***simple_list_item_1.xml*** qui est un simple TextView.

Il suffit de connecter cet adaptateur à un ListView à remplir:

```
ListView listview = (ListView) findViewById(R.id.list);
listview.setAdapter(itemsAdapter);
```

```
<LinearLayout xmlns: …
    android:orientation="vertical" >

    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```

Par défaut, cela convertira désormais chaque élément du tableau de données en une vue en appelant toString sur l'élément, puis en attribuant le résultat comme valeur d'un TextView (simple_list_item_1.xml) qui est affiché comme ligne pour cet élément de données.

```
public class MainActivity extends Activity{
    private String[] modules = new String[]{
         "Applications Mobiles", "Rédaction Scientifique", "Administration BDD",
         "Cryptographie", "Anglais"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ListView mListView = (ListView) findViewById(R.id.list);

        final ArrayAdapter<String> adapter = new ArrayAdapter<String>
             (MainActivity.this, android.R.layout.simple_list_item_1, modules);
        mListView.setAdapter(adapter);
    }
}
```

Si l'application nécessite une traduction plus complexe entre l'élément et View, on peut utiliser plutôt un ArrayAdapter personnalisé.

5.1 Gestion des événements

Pour ajouter un événement de click par exemple, on utilise la méthode :

```
void setOnItemClickListener(AdapterView.OnItemClickListener listener)
```

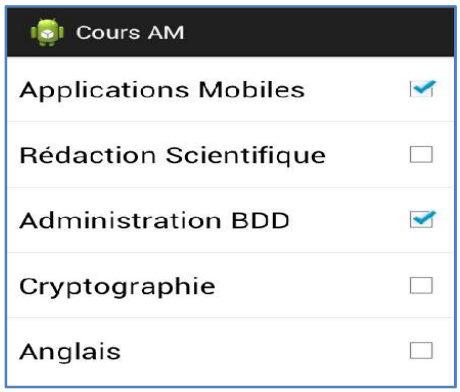
Pour un click simple sur un élément de la liste. La fonction de *callback* associée est :

```
void onItemClick (AdapterView<?> adapter, View v, int pos, long id)
```

- **adapter** l'AdapterView qui contient la vue sur laquelle le click a été effectué,
- **v** c'est la vue en elle-même,
- **pos** c'est la position de la vue dans la liste
- **id** représente est l'identifiant de la vue.

```
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
   @Override
   public void onItemClick(AdapterView<?> adapterView,View view,
                           int position, long id) {
         // faire quelque chose
            }
      });
```

5.2 Sélection des éléments



Dans le cas où l'utilisateur aurait besoin de sélectionner un ou plusieurs éléments, il suffit de spécifier le mode de sélection dans la liste. Ce mode est défini en XML par l'attribut **android:choiceMode** qui peut avoir les valeurs suivantes:

- singleChoice (sélectionner un seul élément)
- multipleChoice (sélectionner plusieurs éléments a la fois)

Dans le cas des sélections multiples, on utilise ***android.R.layout.simple_list_item_multiple_choice*** dans le deuxième paramètre de ArrayAdapter.

Lorsqu'on veut afficher un ensemble d'éléments dans une liste à l'aide d'une représentation personnalisée des éléments, on peut utiliser notre propre disposition XML personnalisée pour chaque élément au lieu de celles proposées par android (par exemple: `android.R.layout.simple_list_item_1`). Pour ce faire, on doit créer notre propre classe ***ArrayAdapter personnalisé***.

6 Conclusion

Ce chapitre prolonge le Chapitre IV sur les interfaces utilisateur, en présentant comment créer des interfaces beaucoup plus riches (menu, dialogue, annonce, et liste) qui offrent des possibilités en ergonomie pour capter l'attention des utilisateurs.

Questions

- Quelle est la différence entre un Menu et un Menu Contextuel.
- Comment peut-on personnaliser une List pour pouvoir sélectionner plusieurs éléments de cette List à la fois.
- Une boite de dialogue peut contenir uniquement du texte -Vrai -Faux

Chapitre VII :

La persistance des données

dans Android

1 Introduction

Android propose quatre (4) techniques pour assurer la persistance des donnes et faire persister des données dans le temps. Chaque technique offre un arrangement entre facilité de mise en œuvre, rapidité et flexibilité de l'application:

- La *persistance des données des activités* : c'est l'enregistrement du parcours de l'utilisateur dans l'application. Quand un utilisateur utilise une application, il est important de garder l'état de l'interface utilisateur pour assurer son retour sur les écrans précédemment utilisés tout on concevant leur états.
- Le *mécanisme de préférence clé/valeur* : les fichiers de préférences sont utilisés pour stocker des préférences utilisateur, la configuration d'une application ou l'état de l'interface d'une activité. Ce mécanisme offre un stockage simple et efficace par la paire clé/valeur de données primitives.
- L'utilisation d'un *système de fichiers* : les fichiers sont le support de stockage primitif pour la lecture et écriture des données dans le système de fichiers d'Android. Les fichiers peuvent être stockés dans un support de stockage interne ou externe d'Android.
- L'utilisation d'une *base de données SQLite* : les bases de données SQLite sont utilisées pour le stockage et la manipulation des données structurées. Cette technique sera détaillée dans le prochain chapitre.

2 La persistance des données des activités

La persistance de l'interface utilisateur est faite grâce aux méthodes du cycle de vie de l'activité. Cela facilite l'enregistrement et la gestion de l'état de l'interface tout au long du cycle de vie de l'activité.

La méthode ***onSaveInstanceState*** d'une activité est appelée pour libérer des ressources ou détruire l'activité (après l'appuie sur la touche *Retour*). L'objet utilisé pour stocker les données est de type ***Bundle***, il est passé comme paramètre aux méthodes ***onCreate*** et ***onRestoreInstanceState*** pour rétablir l'interface utilisateur quand l'activité sera créer ou restaurer respectivement.

Android offre automatiquement la persistance du chemin de navigation de l'utilisateur, qui lui permet de retourner à la bonne activité lorsqu'il appuie sur la touche Retour. Toutefois, le retour vers le parent (bouton "up") n'est pas automatique puisqu'elle doit être personnalisée par le développeur (dans le fichier AndroidManifest avec l'attribut ***android:parentActivityName***) en indiquant l'activité à retourner quand la touche "up" sera appuie.

3 Les préférences partagées (SharedPreferences)

Ce mécanisme permet la persistance de propriétés sous la forme d'un ensemble de paires clé/valeur, il associé un identifiant à une valeur. Cette paire conserve les options que l'utilisateur de l'application souhaite garder ou sauvegarde l'état de l'interface graphique. Ces valeurs peuvent être partagées entre plusieurs composants.

On peut accéder aux SharedPreferences selon 3 manières:

1. La méthode statique
   SharedPreferences PreferenceManager.getDefaultSharedPreferences(Context cxt)

2. Si on veut utiliser un fichier standard par activité, alors on utilise la méthode
   SharedPreferences getPreferences(int mode)
3. Si on a besoin de plusieurs fichiers qu'on les identifiera par leur nom, alors on utilise
   SharedPreferences getSharedPreferences (String name, int mode)

name est le nom du fichier.
mode cette option définit les droits relatives aux fichiers, elle peut prendre trois valeurs :

- Context.MODE_PRIVATE, indique que le fichier créé ne sera accessible que par l'application qui l'a créé.
- Context.MODE_WORLD_READABLE, signifie que le fichier créé peut être lu par n'importe quelle application mais aucune modification n'est possible.
- Context.MODE_WORLD_WRITEABLE, signifie que le fichier créé peut être lu *et* modifié par n'importe quelle application.

3.1 Récupération des préférences partagées

Pour récupérer les préférences on fait un appel à la méthode **getPreferences** de l'activité courante. Cette méthode retourne une instance de la classe **SharedPreferences** à partir de laquelle on peut extraire les valeurs. Une fois l'instance est récupérée, la lecture des données sera effectué à travers les méthodes telles que getBoolean, getString, getFloat, pour récupérer des valeurs de types boolean, String, float respectivement.

Pour chacune de ces méthodes, on spécifié le nom de la clé et la valeur à fournir par défaut si la valeur correspondant à la clé n'existe pas.

```
public static final String NOM_PREFS = "MesPreferences";

  ...

SharedPreferences mesPrefs = getSharedPreferences(NOM_PREFS, Context.MODE_PRIVATE);
           int ident = mesPrefs.getInt("id", 0);

               String nom = prefs.getString("nomUtilisateur", null);
```

Si on veut extraire toutes les paires clé/valeur des préférences en une seule opération, on utilise la méthode **getAll** qui permet de retourner un objet de type **Map<String, ?> .**

Les préférences sont enregistrées dans des fichiers du système de fichiers interne d'Android. Si on prend notre exemple précédent, le fichier est enregistré à l'emplacement suivant :

/data/data/<package de l'application>/shared_prefs/MesPreferences.xml

3.2 Mise à jour des préférences partagées

L'interface **Editor** de la classe SharedPreferences permet d'enregistrer les préférences et offre toutes les méthodes de mise à jour (ajouter, modifier, suppression) des valeurs.

Pour récupérer une instance de la classe Editor, on fait appel à la méthode **edit** de l'instance SharedPreferences, et on utilise les méthodes putBoolean, putString, putInt, putFloat et putLong pour ajouter des valeurs respectivement de type String, Int, Float et Long.

Si une valeur est spécifiée avec une clé qui existe déjà, la valeur sera simplement mise à jour. Lorsque les modifications sont apportées, la méthode **commit** de l'objet Editor est appelée pour enregistrer les données.

```
SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);
  // Récupération d'un éditeur pour modifier les préférences.
 SharedPreferences.Editor editor = prefs.edit();
 editor.putString("nomUtilisateur", "ALI");
 editor.commit();
```

La méthode remove(key) permet de supprimer une préférence, et pour supprimer toutes les préférences on peut utiliser la méthode *clear* de l'interface editor. Par exemple : `editor.clear();`

3.3 Gestion des événements

La classe SharedPreferences dispose de la méthode **registerOnSharedPreferenceChangeListener** qui permet de désigner une méthode de rappel/écouteur qui sera appelée à chaque mise à jour des préférences.

```
prefs.registerOnSharedPreferenceChangeListener(
                new OnSharedPreferenceChangeListener() {
  @Override
  public void onSharedPreferenceChanged(
        SharedPreferences sharedPreferences, String key) {
          // faire quelque chose
      }
 );
```

3.4 Activité de préférences

Pour afficher un écran d'édition des préférences relative à sa description XML, on doit créer une nouvelle activité héritant de la classe **PreferenceActivity** et solliciter la méthode **addPreferencesFromResource** en fournissant l'id de la description XML:

```
public class MyPrefs extends PreferenceActivity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);
      }}
```

La classe PreferenceActivity offre plusieurs façons pour construire l'interface de préférences:

- Par un fichier XML que est placé dans les ressources de projet (/res/xml/preferences.xml)
- En utilisant des activités pour lesquelles on a spécifié des métadonnées dans le fichier AndroidManifest.xml ;
- A partir d'une hiérarchie d'objets dont la racine est de type PreferenceScreen.

3.5 Les écrans de préférences

Consiste à créer un fichier XML décrivant comment sera présenté l'écran de préférences à l'utilisateur. Les attributs suivants sont très utiles:

- **android:title**: le nom de la préférence
- **android:summary**: une phrase qui permet d'expliquer le rôle de la préférence
- **android:key**: c'est la clé d'enregistrement de la préférence. C'est un attribut *indispensable*.
- **android:defaultValue** : c'est la valeur par défaut de la préférence.
- **android:dependency** : permet d'établir une liaison avec une autre préférence via la clé de cette dernière. Pour cela, on doit lui attribuer l'identifiant android:key de la préférence dont il dépend.

La méthode **getDefaultSharedPreferences** de la classe **PreferenceManager** peut être aussi utilisée pour accéder aux valeurs des préférences à travers la clé indiquée par l'attribut **android:key** qui est utilisée pour récupérer la valeur choisie par l'utilisateur.

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(
                                                    getApplicationContext());
String nom = prefs.getString("nomUtilisateur ","");
```

```
<PreferenceScreen  xmlns:android="http://schemas.android.com/apk/res/android"
                   android:key="premier_ecran">
       <CheckBoxPreference     android:key="wifi enabled"
                               android:title="WiFi" />
       <PreferenceScreen       android:key="deuxieme_ecran"
                                android:title="Configuration WiFi">
           <CheckBoxPreference      android:key="prefer wifi"
                                   android:title="WiFi Préférer" />
                      //  ... other preferences here ...
       </PreferenceScreen>
</PreferenceScreen>
```

3.6 Les types de préférences

CheckBoxPreference

C'est une case à cocher avec deux valeurs *true* ou *false*, c'est-à-dire soit la case est cochée, soit non. L'attribut **android:summaryOn** permet de définir un résumé qui ne s'affiche que lorsque la case est cochée, par contre l'attribut **android:summaryOff** permet définir un autre résumé mais seulement lorsque la case est décochée.

```
<CheckBoxPreference
        android:key="checkBoxPref"
        android:title="Titre"
        android:summaryOn="Résumé quand sélectionné"
        android:summaryOff="Résumé quand pas sélectionné"
        android:defaultValue="true"/>
```

EditTextPreference

Permet d'ouvrir une boîte de dialogue qui contient un EditText permettant à l'utilisateur d'insérer du texte. L'attribut **android:dialogTitle** définie le texte de la boîte de dialogue, les deux attributs **android:negativeButtonText** et **android:positiveButtonText** définissent respectivement le texte du bouton à droite et du bouton à gauche de la boîte de dialogue.

```
<EditTextPreference
    android:key="editTextPref"
    android:dialogTitle="Titre de la boîte"
    android:positiveButtonText="OK !"
    android:negativeButtonText="Annuler !"
    android:title="Titre"
    android:summary="Résumé"
    />
```

4 Stockage dans des fichiers

Afin de simplifier la manipulation des fichiers dans un contexte local, Android fournit les deux méthodes **openFileOutput** et o**penFileInput**.

La méthode openFileOutput ouvre un fichier en mode écriture. Pour insérer des données dans ce fichier, il faut spécifier le mode MODE_APPEND sinon le fichier sera replacer (écraser). Si le fichier n'existe pas, il sera crée.

```
private static final String NOM_FICHIER = "MonFichier.dat";
    ...
     try {
       // Crée un flux de sortie vers un fichier local.
       FileOutputStream ous = openFileOutput(NOM_FICHIER, MODE_PRIVATE);
        ... ou
      // Crée un flux d'entrée à partir d'un fichier local.
       FileInputStream ins = openFileInput(NOM_FICHIER);

     } catch (FileNotFoundException e) {
      // Traitement des exceptions ...
      }
```

Ces méthodes prennent en charge les noms de fichiers qui se trouvent uniquement dans le répertoire de l'application.

Pour supprimer un fichier, Android propose la méthode **deleteFile** qui permet d'effacer un fichier à partir de son nom.

Android propose aussi des méthodes pour manipuler les fichiers depuis le contexte de l'application :

- **fileList** permet d'afficher tous les fichiers locaux de l'application.
- **getFileDir** retourne le chemin absolu du répertoire qui contient tous les fichiers créés par la méthode openFileOutput.
- **getFileStreamStore** retourne le chemin absolu du répertoire qui contiens le fichier créé avec openFileOutput et dont le nom est passé en paramètre.

5 SQLite

SQLite est un moteur de base de données relationnel open source. Il est naturellement portable puisque le moteur et les données sont contenus dans un même fichier. Il permet de créer des BDD légères internes aux applications Android. SQLite ne requiert pas de serveur pour fonctionner, cela veut dire que son exécution se fait dans le même processus que celui de l'application. Donc, c'est un excellent choix pour le stockage de données dans un smartphone.

L'interaction avec une base de données SQLite peut être faite selon différent manières grâce à trois (3) classes :

5.1 La classe SQLiteDatabase

SQLiteDatabase est la classe de base qui permet l'interaction avec une base de données SQLite sous Android et propose des méthodes pour ouvrir/fermer, exécuter des requêtes, mettre à jour la base de données. SQLiteDatabase fournit également la méthode **execSQL ()** qui permet d'effectuer des instructions SQL directement.

L'objet **ContentValues** définie les paires *clés* / *valeurs*. La clé est l'identifiant de la colonne de la table. La valeur est le contenu de l'enregistrement dans cette colonne. La méthode ContentValues est utilisée aussi pour mettre à jour les enregistrements de la base de données.

Création de BDD

Pour créer une base de données, il vous suffit d'appeler la méthode **openOrCreateDatabase** avec *le nom* de la base de données à créée et *le mode* comme paramètre. Elle retourne une instance de la base de données SQLite.

```
SQLiteDatabase bdd = openOrCreateDatabase("nom_bdd",MODE_PRIVATE,null);
```

Les bases de données sont stockées dans les répertoires sous la forme suivante : /data/data/<package>/databases.

Une application peut contenir plusieurs bases de données au même, mais chaque base de données est créé selon le mode MODE_PRIVATE, par conséquent elles ne sont accessibles qua partir de leur application.

La classe SQLiteDatabase propose 3 approches qui permettent d'exécuter des requêtes SQL sur une base de données SQLite :

5.1.1 Approche par SQL brut

Pour exécuter une requête SQL sans avoir besoin de réponse, il suffit d'utiliser la méthode:

```
void execSQL(String sql)
```

Création de table

La création de table se fait avec la syntaxe suivante:

```
CREATE TABLE nom_de_la_table (
   nom_du_champ_1 type {contraintes},
   nom_du_champ_2 type {contraintes},
                          …);
```

If faut déclarer au moins les deux informations suivantes pour chaque attribut:

- Le nom, afin de pouvoir l'identifier
- Le type de donnée.

```
bdd.execSQL( "CREATE TABLE  Personne (" +"id INTEGER PRIMARY KEY AUTOINCREMENT,"
                                        +"nom TEXT NOT NULL," +"age INTEGER) ");
```

SQLite support uniquement 5 ***types*** de données :

- *NULL* pour les données Nulles.
- *INTEGER* pour les entiers.
- *REAL* pour les nombres réels.
- *TEXT* pour les chaînes de caractères.
- *BLOB* pour les données brutes (une image par exemple).

SQLite peut gérer ***des contraintes*** sur une ou plusieurs colonnes. Parmi les contraintes on peut citer :

- PRIMARY KEY pour attribuer une clé primaire sur un attribut.
- NOT NULL pour préciser que cet attribut ne peut avoir la valeur NULL.
- CHECK permet de vérifier que la valeur de cet attribut est cohérente
- DEFAULT permet de définir une valeur par défaut.
- UNIQUE crée automatiquement un index sur la ou les colonnes auxquelles elle est attribuée.
- AUTOINCREMENT : permet à un numéro unique (souvent le champ PRIMARY KEY) d'être généré automatiquement lorsqu'un nouvel enregistrement est inséré dans une table.
- FOREIGN KEY : chaque colonne référencée par une clé étrangère doit être déclarée comme UNIQUE.

Insertion / Suppression

Pour insérer des données dans la table ou supprimer une table on utilise la méthode **execSQL** définie dans la classe SQLiteDatabase.

```
bdd.execSQL("INSERT INTO Personne VALUES('Mohamed','21');");
dbb.execSQL("DROP TABLE Personne ";");
```

Recherche et sélection

Pour récupérer des le contenu de la base de données on peut utiliser l'objet de la classe **Cursor**. On appel la méthode **rawQuery** qui retourne un ensemble de résultats avec le curseur pointant sur la table. On peut déplacer le curseur vers l'avant et récupérer les données.

```
Cursor resultSet = bdd.rawQuery("SELECT * FROM Personne ", null);
    resultSet.moveToFirst();
    String nom = resultSet.getString(1);
       Int age = resultSet.getInt(2);
```

5.1.2 Approche par composants

Android offre des méthodes spécifiques pour insérer, modifier, supprimer des n-uplets. La différence avec **execSQL**, c'est que ces méthodes requièrent un tableau de String, et par conséquent, on doit convertir toutes les données en chaînes de caractères.

Insertion
La syntaxe suivante permet d'ajouter une entrée dans la table:

`insert(table, null, valeurs)` qui effectue une requête de type :

```
INSERT INTO nom_de_la_table (colonne1, colonne2,…) VALUES (valeur1, valeur2, …)
```

La méthode retourne l'identifiant du nouveau n-uplet (la ligne). Ses paramètres sont :

- table : fournir le nom de la table
- null sauf si on veut INSERT INTO table sans fournir aucune valeur.
- valeurs : c'est une structure du type ContentValues qui associe des noms et des valeurs.

Exemple

```
ContentValues valeurs = new ContentValues();
              valeurs.putNull("id");
              valeurs.put("nom", "Mohamed");
              valeurs.put("age", "21");
 int id = bdd.insert("personnes", null, valeurs);
```

Suppression

La méthode utilisée pour supprimer est:

```
int delete(String table, String whereClause, String[] whereArgs)
```

Cette méthode retourne un entier qui contient le nombre de lignes supprimées. Les paramètres sont:

- table : fournir le nom de la table
- whereClause : une condition contenant des jokers **?**
- whereArgs : est un tableau des valeurs à mettre à la place des **?**

```
bdd.delete("personnes ","id BETWEEN ? AND ?", new String[]{"1","3"});
```

Modification

```
int update(String table, ContentValues values, String whereClause,
                                                String[] whereArgs).
```

- Pour effectuer des modifications sur le ou les enregistrements cibles, on peut ajouter juste le paramètre *values* (ce sont les couples (colonne, valeur à mettre)). En

générale, si on veut mettre à jour un champ d'une table, il suffit de mettre à jour l'objet associé et d'insérer la nouvelle valeur dans un ContentValues.

```
ContentValues valeurs = new ContentValues();
              valeurs.put("nom", "Mohamed");
      bdd.update("personnes ", valeurs, "id=?", new String[] { "4" });
```

5.1.3 Approche par modèle (patron)

Ce principe consiste à définir une classe pour chaque table. Cette classe regroupe toutes les requêtes SQL qui la concernant sous forme de méthodes de classe: création, suppression, mise à jour, parcours, insertions, etc. Les instances des classes représentent les n-uplets (les lignes) de la table. Toutes les méthodes de ces classes prennent la base de données comme premier paramètre.

Par exemple, la méthode *find* permet de récupérer les n-uplets, et les méthodes *new*, *save* et *delete* permettent la mise à jour de la base.

Par exemple: On définit une classe qui contient que des méthodes statiques pour créer et supprimer une table :

```
public static void create(SQLiteDatabase bdd){

    bdd.execSQL( "CREATE TABLE Personne (" +"id INTEGER PRIMARY KEY AUTOINCREMENT,"
                                          +"nom TEXT NOT NULL," +"age INTEGER) ");
           }
public static void drop(SQLiteDatabase bdd){
    bdd.execSQL("DROP TABLE IF EXISTS Personne");
           }
```

Les méthodes qui permettent de créer ou supprimer des n-uplets (*Perso* est une classe java qui contient les méthodes (getters et setters) de la table Personne) sont les suivantes:

```
public static void insert(SQLiteDatabase bdd, Perso pr) {

         bdd.execSQL( "INSERT INTO Personne VALUES (null, ?, ?)",
                       new Object[] { pr.getNom(), pr.getAge() });
}
public static void delete(SQLiteDatabase bdd, long id){
        bdd.execSQL(" DELETE FROM Personne WHERE id=? ", new Object[] {id});
 }
```

5.2 La classe Helper

Android a des fonctionnalités disponibles pour manipuler des schémas de base de données en constante évolution, qui dépendent principalement de l'utilisation de la classe **SQLiteOpenHelper**.

SQLiteOpenHelper est conçu pour faire face a de deux problèmes très courants.

- Lorsque l'application s'exécute pour la première fois. À ce stade, on n a pas encore

de base de données. Donc, on doit créer les tables, les index, les données de démarrage, etc.

- Lorsque l'application est mise à niveau vers un schéma plus récent. Notre base de données sera toujours sur l'ancien schéma de l'ancienne édition de l'application. On aura la possibilité de modifier le schéma de la base de données pour répondre aux besoins du reste de l'application.

SQLiteOpenHelper conclut ces logiques pour créer et mettre à niveau une base de données selon nos spécifications. Pour cela, on doit créer une sous-classe personnalisée de SQLiteOpenHelper. Dans le constructeur cet cette classe, on fait appel à la méthode ***super()*** de SQLiteOpenHelper, en précisant le nom de la base de données et sa version actuelle, et on doit implémenter au moins les deux méthodes suivantes:

- **onCreate(SQLiteDatabase db)** est appelée pour accéder à une base de données qui n'est pas encore créée. Elle retourne un objet **SQLiteDatabase** qui pointe vers une base de données nouvellement créée, qu'on peut le remplir avec des tables et des données initiales.

- **onUpgrade**(SQLiteDatabase db, int oldVersion, int newVersion): elle est appelée lorsque la version de schéma dont on a besoin ne correspond pas à la version de schéma de la base de données. Elle renvoie un objet SQLiteDatabase avec les anciens et nouveaux numéros de version. Cette méthode permet de modifier un schéma de base de données existant ou de supprimer la base de données existante et la reconstruire par la méthode onCreate(). C'est la meilleure façon de convertir la base de données de l'ancien schéma vers le nouveau. Android vérifie les numéros des versions actuelle et précédente de la base de données à chaque lancement. Si le numéro de la version actuelle est supérieur à celui de la précédente, alors il fait appel à cette méthode.

```
public class DatabaseOpenHelper extends SQLiteOpenHelper {

private static final String createmaTable = "CREATE TABLE maTable " + " ( ID " + "
                                    INTEGER PRIMARY KEY," + " Title TEXT " + " )";

public static final int databaseVersion = 1;
public static final String databaseName = "maBDD";
private static final String deleteTable = "DROP TABLE IF EXISTS maTable" ;

   public DatabaseOpenHelper(Context context) {
         super(context, databaseName, null, databaseVersion);
       }
   @Override
   public void onCreate(SQLiteDatabase db) {
         db.execSQL(createmaTable);
                     }
   @Override

   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
         db.execSQL("DROP TABLE " + maTable + ";");
         onCreate(db);
                    }
 }
```

Les deux méthodes prennent en paramètre un objet SQLiteDatabase qui représente une instance Java de la BDD. La classe SQLiteOpenHelper propose les méthodes **getReadableDatabase()** et **getWriteableDatabase()** pour accéder à un objet SQLiteDatabase en lecture et en écriture respectivement.

5.3 La classe Curseur

Un curseur est un objet qui retourne le résultat d'une requête et pointe vers une ligne de ce résultat. Il permet à Android de gérer les résultats des requêtes de façon efficace, puisqu' il évite le chargement de toutes les données en mémoire.

Parmi les méthodes de Cursor :

- *getCount ()* permet de retourner le nombre d'éléments résultant de la requête.
- *MoveToFirst ()* et *MoveToNext ()* pour se déplacer entre les lignes individuelles de données
- *isAfterLast ()* cette méthode retourne un booléen, cette valeur est vraie, si le curseur se trouve après la fin du résultat de la requête, sinon il retourne faux.

Le curseur fournit aussi des méthodes get**X**() avec **X** : type de données. Par exemple: les méthodes get**Long** (columnIndex), get**String** (columnIndex) permettent d'accéder aux données d'une colonne de la position courante du résultat. Le "columnIndex" est l'index de la colonne de résultat.

Le curseur propose aussi la méthode getColumnIndexOrThrow (String) la méthode getColumnIndexOrThrow (String) prend en paramètre le nom d'une colonne et retourne l'index de cette colonne. L'appel de la méthode close () permet de fermé le curseur.

Le pseudo code suivant permet de naviguer entre toutes les lignes d'un curseur:

```
while (cursor.moveToNext()) {
   // Faire quelque chose
  }
 cursor.close();
```

La méthode **rawQuery** renvoie un objet de type Cursor permettant de ***parcourir les n-uplets un par un*** :

```
Cursor cursor = bdd.rawQuery("SELECT * FROM Personne WHERE...");
   if (cursor != null) {
     try {
       if (cursor.moveToFirst()) {
           while (!cursor.isAfterLast()) {
              // faire quelque chose
           cursor.moveToNext();
                                          }
                                    }
        } finally {
      cursor.close();
                        }
```

Pour récupérer les données, la méthode la plus simple a utilisé (~ SELECT) est:

public Cursor query (boolean **distinct**, String table, String[] **columns**, String **whereClause**,

String[] **selectionArgs**, String **groupBy**, String **having**, String **orderBy**) ;

- distinct permet d'éviter la redondance des résultats.
- columns est la liste des colonnes à retourner. Pour avoir toutes les colonnes on met la valeur *null*.
- whereClause est la clause WHERE d'un SELECT (sans le mot WHERE). Pour retourner toutes les lignes on met la valeur *null*.
- selectionArgs est utile lorsque whereClause contiens des paramètres notés **?** Les valeurs de ces paramètres sont désignées par selectionArgs.
- groupBy est la clause GROUP BY d'un SELECT. Utile pour des SELECT COUNT(*).
- having indique les groupes de lignes à retourner (comme HAVING de SQL = un WHERE sur résultat d'un calcul, pas sur les données)
- orderBy est la clause ORDER BY d'un SELECT. Pour ne pas tenir compte de l'ordre, on met *null*.

6 Conclusion

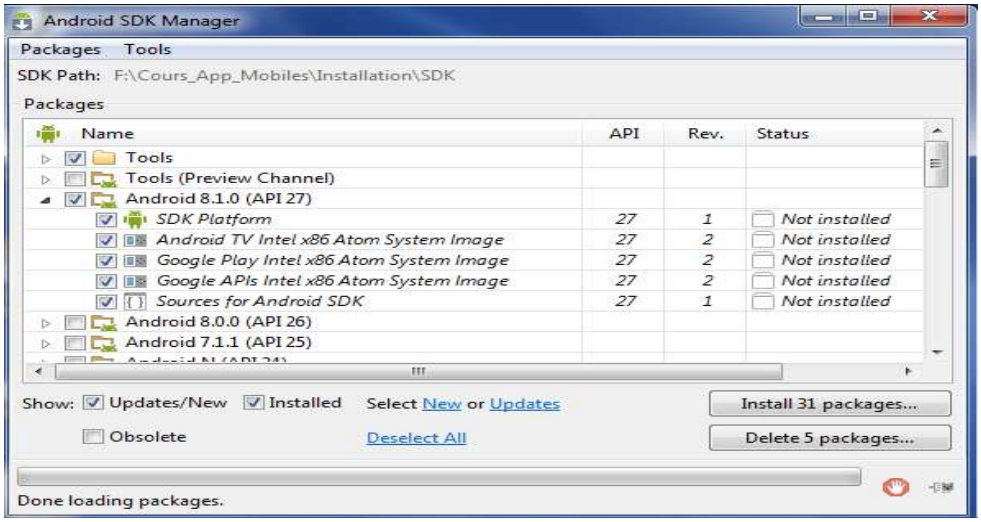
Dans ce chapitre, nous avons présenté un aperçu sur les différents mécanismes utilisés par Android pour stocker, manipuler et accéder aux données depuis une application mobile.

TP 1 (Installation et configuration Android)

Installations

Pour préparer l'environnement de développement Android, nous avons besoin de télécharger et d'installer les outils suivants :

- **JDK**[16] : Le kit de développement Java (s'il n'est pas déjà installer sur votre machine)
- **Android SDK**[17] : Télécharger le kit de développement Android fourni par Google dans le lien suivant : Télécharger le Android SDK. Une fois le fichier archive (.zip) est récupéré, décompressez-le dans votre répertoire d'installation (par exemple : C:\android_env).

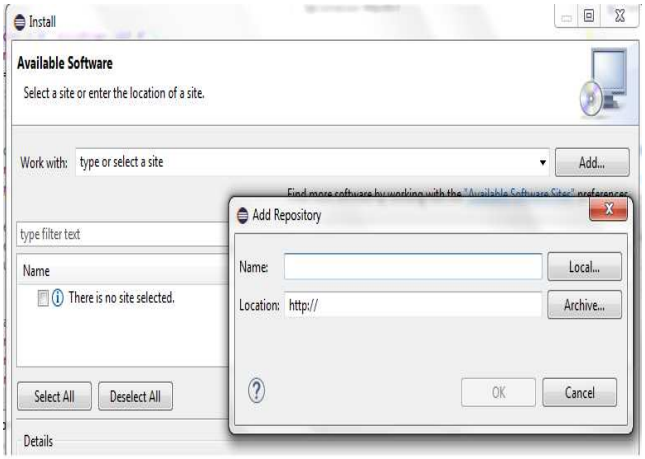


Dans le répertoire de votre SDK téléchargé, vous lancez (double-clique) le *SDKManager.exe*, et sélectionnez les outils (Tools), et les API Android, puis cliquez sur *Install*. Il faut mentionner que le choix de l'API dépend des capacités de votre machine. Si vous disposez d'une mémoire RAM inférieure au égale à 4 Go, il vaut mieux choisir une API inferieur à 20 (API Level<20).

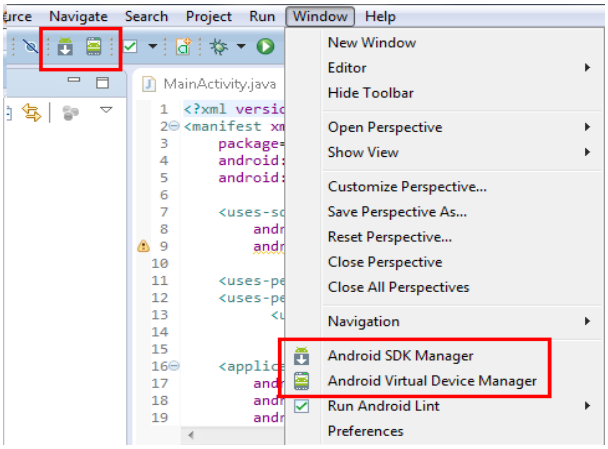
- **Eclipse**[18] : après le teste de plusieurs versions d'Eclipse, nous avons constaté que la version Luna est la mieux adaptée pour fonctionner avec le plugin ADT version 23.0.4.
- **Plugin ADT** :
  1. Démarrez Eclipse, et cliquer sur "Help" dans le menu principal, puis "Install New Software".
  2. Cliquez sur le bouton "Add…"
  3. Dans le premier champ, entrez un nom pour le ADT "Android Developer Tools" puis dans le second champ :
     - Saisissez "**https://dl-ssl.google.com/android/eclipse/**". Validez par un clic sur le bouton "OK".
     - Si vous ne disposez pas de connexion Internet, vous cliquez sur le bouton «archive », puis sélectionnez l'archive de plugin ADT dans votre disque (Exemple : c:\ADT-23.0.4.zip).

[16] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[17] https://developer.android.com/studio/index.html
[18] https://www.eclipse.org/luna/

4. Sélectionnez la case à cocher "Developer Tools", puis cliquez sur Next.
5. Dans la fenêtre suivante, vous verrez une liste des outils à télécharger. Cliquez sur Next, acceptez la License d'utilisation, puis cliquez sur "Finish". Eclipse télécharge ensuite les composants.
6. Redémarrez Eclipse pour terminer l'installation du plugin.



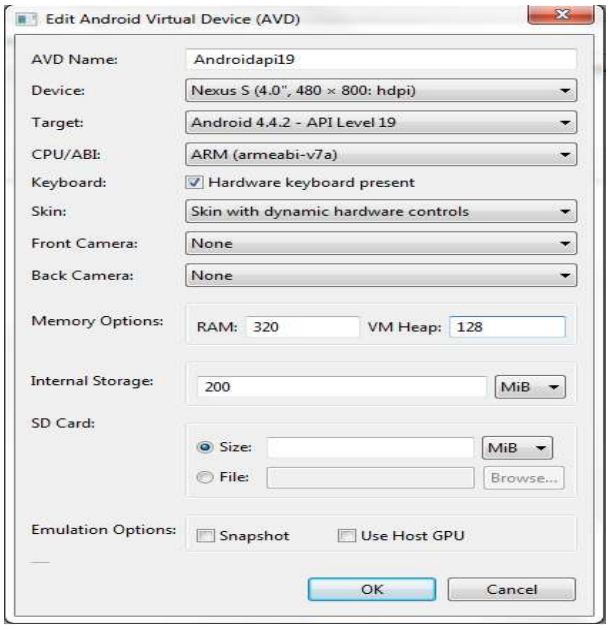
Les étape 1, 2 et 3



Après l'étape 6

PS. Nous avons opté pour cette configuration (Eclipse + ADT), car elle est la mieux adaptée aux capacités (RAM = 2Go et CPU : Duel Core) des machines dans nos salles de TP.

Si vous avez un PC avec RAM > 4Go, alors vous pouvez installez **Android Studio** au lieu de Eclipse, car c'est l'IDE le plus adapté actuellement pour le développement des applications Android.



Configuration d'Emulateur

Pour créer un émulateur AVD, vous lancez d'abord le *AVDManager*, soit directement à partir de votre répertoire AndroidSDK ou à travers le bouton AVDManger dans Eclipse, puis dans l'onglet Android Virtuel Devices vous tapez le bouton *Create.* En suite :

- Choisissez un nom significatif pour votre périphérique virtuel.
- Dans **Target**, sélectionnez un SDK Android, la liste déroulante contient vos versions installées du SDK Android.
- Dans **Device**, sélectionnez le périphérique Android à émuler.
- Dans **Memory Options**, donnez la taille de la RAM et VM Heap en Mo.

Conseil : L'émulation de **Nexus S** avec Android4.4.2 (**API level19**), RAM=**320Mo**, VM Heap=**128Mo** et Internal Storage= **200Mo** peut être plus rapide que l'émulation d'un autre périphérique plus récent. Cette configuration AVD est la plus adéquate pour nos machines de TPs (Faculté FSEI, Univ. Mosta).

TP2 (Interface Graphique)

Exercice 1 : Réaliser l'interface suivante (Fig.1) en utilisant un LinearLayout et on prenant en considération les points suivant :

- La chaine de caractère qui est affichée dans le composant EditText (Entrer le login) disparaisse quand vous positionnez le curseur sur ce composant.
- Les deux boutons partagent équitablement la largeur de l'écran.

Fig.1

Exercice 2 : Modifiez l'interface réalisée dans l'exercic1 pour que :

1. Les deux boutons doivent être positionnés au centre de l'espace restant de l'écran comme il est montré dans la **Fig.2**
2. Tous les composants de l'interface graphique se positionnent au centre de l'écran comme le montre la **Fig.3**

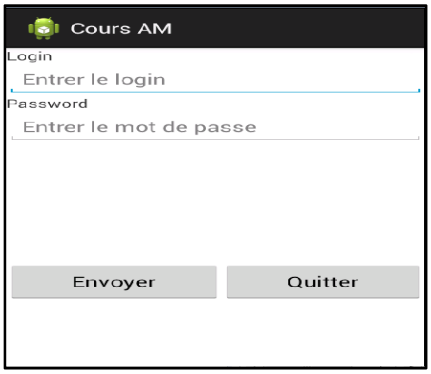


Fig.2

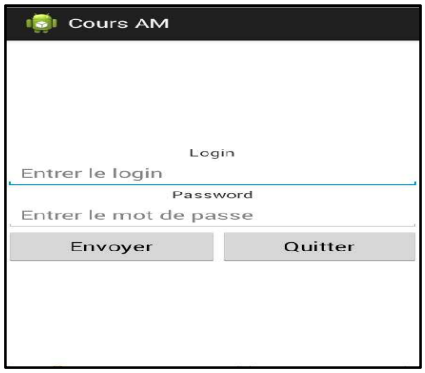


Fig.3

Exercice 3 : Modifiez l'interface réalisée dans l'exercic1 pour qu'elle ressemble à celle de **Fig.4.** Vous pouvez utiliser les codes de couleur suivants : **#81D8D0** **#FFFF00**

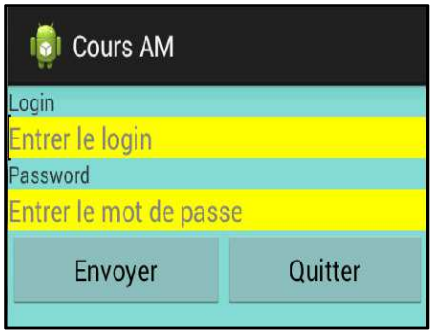


Fig.4

TP3 (communication entre composants)

La fenêtre **LogCat** dans Android affiche les messages du système. Pour afficher la fenêtre de **LogCat** dans éclipse : Allez dans la barre de menu d'ecplise cliqué sur Windows--> Show View--> Other--> Android--> LogCat.

La classe ***android.util.Log*** propose plusieurs méthodes de trace (de Log) hiérarchisées. Ces méthodes ont pour nom une seule lettre comme type de message.

- **Verbose v :** affiche tous les messages du journal (par défaut).
- **Debug d :** affiche les messages du journal de débogage qui sont utiles uniquement pendant le développement.
- **Info i :** affiche les messages de journal attendus pour une utilisation régulière, ainsi que les niveaux de message plus bas dans cette liste.
- **Warn w :** affiche les problèmes possibles qui ne sont pas encore des erreurs, ainsi que les niveaux de message inférieurs dans cette liste.
- **Error e :** afficher les problèmes qui ont causé des erreurs, ainsi que le niveau de message inférieur dans cette liste.

Ces méthodes ont deux arguments : **Log**.*TypedeMessage*(String tag, String msg)

Elles permettent de visualiser des traces lors de l'exécution en utilisant l'onglet **LogCat**. On peut filtrer ces traces en ne laissant afficher que les logs de balise **tag**

Par exemple : `Log.v("Cours AM", "Mon message tracer");`

Exercice 1 :

Utiliser l'interface conçue dans la fiche TP précédente (voir Fig.1) pour ajouter un événement de **click** pour les boutons *Envoyer* et *Quitter.*

Fig.1

Fig.2

- Afficher le contenu du champ de texte (login) comme un message dans le **LogCat** en cliquant sur le bouton *Envoyer*.
- Afficher la chaine de caractère `"Quitter l'application"` en cliquant sur le bouton *Quitter*.

Exercice 2 :

Dans le même projet de votre application, créer une nouvelle activité avec une nouvelle interface (layout) comme illustré dans **Fig.2**. Cette nouvelle interface utilise *RelativeLayout*.

- Modifier le click du bouton ***Envoyer*** pour qu'il puisse lancer la Nouvelle Activité.
- Ajouter un événement de click pour le bouton *Retour* de la Nouvelle Activité. Ce click permet le retour à l'activité principale.

Exercice 3 :

Afficher le contenu des champs de texte *Login* et *Password* de l'activité principale dans la Nouvelle Activité. Vous pouvez utiliser deux (2) nouveaux composants *TextView* pour les afficher dans la Nouvelle Activité.

TP4 (Liste, Dialogue, Menu)

Exercice 1 : Réaliser l'interface présentée dans Fig.1, cette interface affiche une liste de modules. Le bouton « Ajouter » permet de lancer une boite de dialogue qui contient un champ de saisie de texte et deux (2) boutons (voir Fig.2). Quand un nouveau module est ajouté, la liste sera mise à jour automatiquement comme le monte Fig.3.

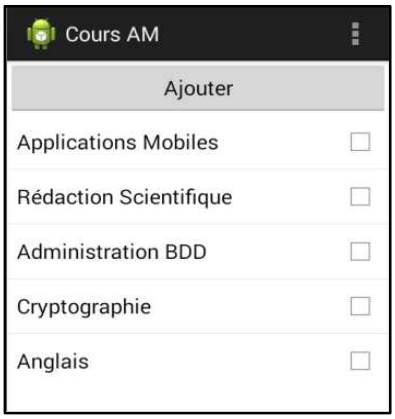


Fig.1

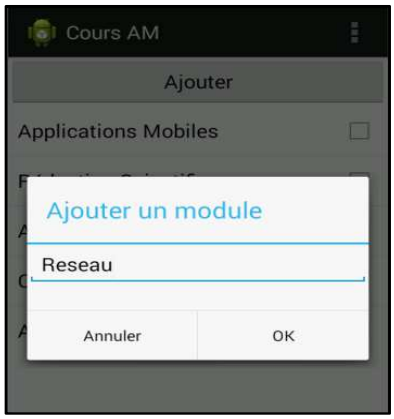


Fig.2

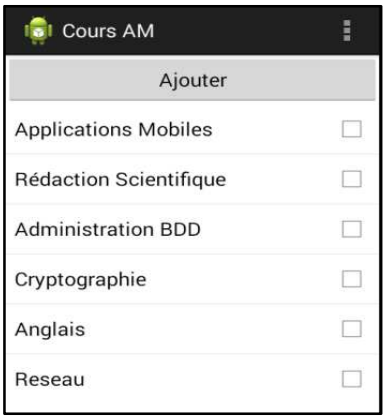


Fig.3

NB. La méthode ***notifyDataSetChanged( )*** permet de maitre a jour l'adaptateur attaché à la liste.

Exercice 2 : Dans la même application, ajouter un menu qui contient 3 items (éléments) comme le montre **Fig.4**. L'item « ajouter » fait appel à la boite du dialogue qui permet d'insérer un nouveau module.

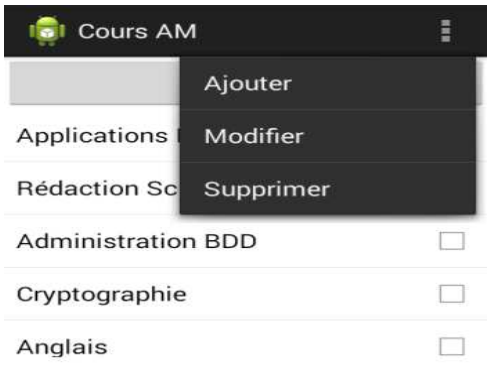


Fig.4

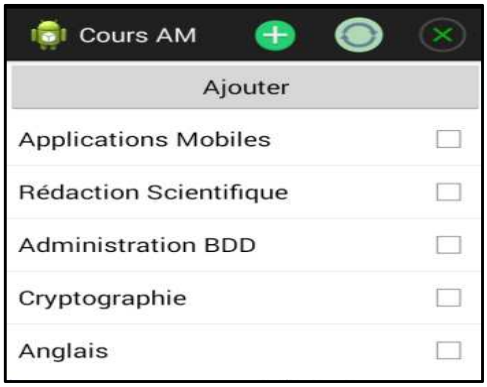


Fig.5

- Modifier les attributs du menu pour qu'il s'affiche dans la barre de titre comme illustré dans **Fig.5**
- Ajouter une action de click pour les éléments de la liste qui permet d'afficher, dans une boite de dialogue, l'indice de l'élément sélectionné.

NB. Vous pouvez utiliser l'icône de Android située dans res/drawable.

Solution TP2 (Interface Graphique)

Exercice 1

```
<?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" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Login" />
    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Entrer le login" />
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Password" />
    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Entrer le mot de passe" />

<LinearLayout
    android:layout_width= "match_parent"
    android:layout_height="wrap_content"
    android:weightSum="100"
    android:orientation="horizontal" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Envoyer"
        android:layout_weight="50" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Quitter"
        android:layout_weight="50" />

  </LinearLayout>
</LinearLayout>
```

- Pour afficher une chaine de caractère dans le composant EditText qui disparaisse quand on positionne le curseur sur ce composant, on utilise l'attribut **android:hint=" … "**

- Pour partager équitablement la largeur de l'écran sur les 2 boutons, on utilise l'attribut **android:weightSum = *"100"*** dans le layout qui contient les deux boutons (c-a-d 100% de l'espace layout). On ajoute l'attribut

android:layout_weight *"50"* dans chaque bouton. Chaque bouton prend 50% de l'espace de son conteneur (layout).

Exercice 2 : Modifiez l'interface réalisée dans l'exercic1 pour que :

3. Les deux boutons doivent être positionnés au centre de l'espace restant de l'écran comme il est montré dans la **Fig.2**
4. Tous les composants de l'interface graphique se positionnent au centre de l'écran comme le montre la **Fig.3**

```
<LinearLayout …   >
    <TextView  …    />
    <EditText  …    />
    <TextView  …    />
    <EditText  …    />

<LinearLayout
    android:layout_width= "match_parent"
    android:layout_height= "match_parent"
    android:weightSum="100"
    android:orientation="horizontal"
    android:gravity="center"  >

    <Button …  />
    <Button …  />

  </LinearLayout>
</LinearLayout>
```

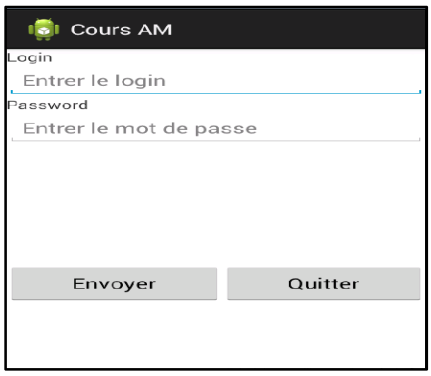


Fig.2

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"  >
    <TextView  …    />
    <EditText  …    />
    <TextView  …    />
    <EditText  …    />

<LinearLayout
    android:layout_width= "match_parent"
    android:layout_height= "wrap_content"
    android:weightSum="100"
    android:orientation="horizontal" >

    <Button …  />
    <Button …  />

  </LinearLayout>
</LinearLayout>
```

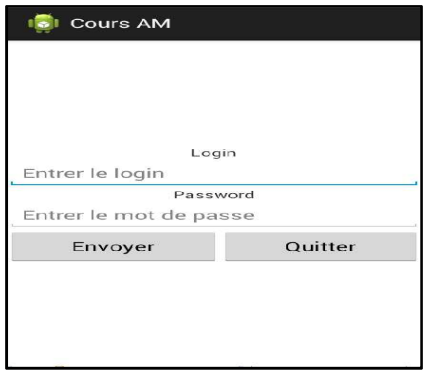


Fig.3

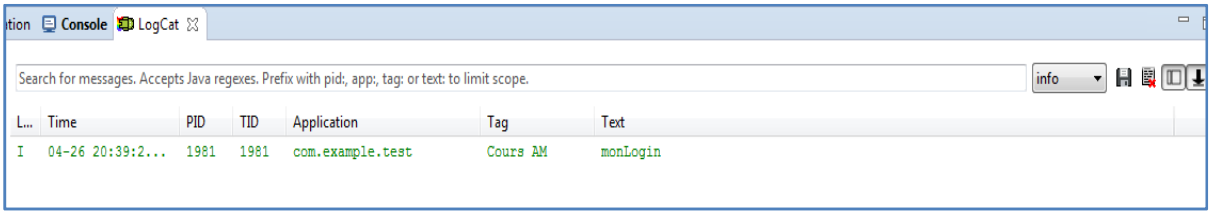
Exercice 3 : Utilisez le même code de l'exercice 1 avec les modifications suivantes :

- Dans le LinearLayout on ajoute l'attribut : `android:background="#81D8D0"`
- Dans les composants EditText on ajoute l'attribut : `android:background="#FFFF00"`

Solution TP3 (communication entre composants)

Exercice 1 :

- Afficher le contenu du champ de texte (login) comme un message dans le **LogCat** en cliquant sur le bouton *Envoyer*.



- Afficher la chaine de caractère `"Quitter l'application"` en cliquant sur le bouton *Quitter*.

```
import android.util.Log;
...

public class MainActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {

     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);

     Button envoyer = (Button) findViewById(R.id.button1);
     Button quitter = (Button) findViewById(R.id.button2);

       envoyer.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
           Log.v("Cours AM ", et.getText().toString());
                    }
              });

       quitter.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
           Log.v("Cours AM ", "Quitter l'application");
                    }
              });

    }
  ...
}
```

Exercice 2 :

Dans le même projet de votre application, créer une nouvelle activité avec une nouvelle interface (layout) comme illustré dans **Fig.2**. Cette nouvelle interface utilise *RelativeLayout*.

1. ***Coté interface*** (activity_secondactivity.xml)

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/andro
id"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FFFF00"  >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="94dp"
        android:layout_marginTop="38dp"
        android:text="Nouvelle Activité" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_alignRight="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:text="Retour" />

</RelativeLayout>
```

Fig.2

- Ajouter un événement de click pour le bouton *Retour* de la Nouvelle Activité. Ce click permet le retour à l'activité principale.

2. **Coté code java** (SecondActivity.java) :

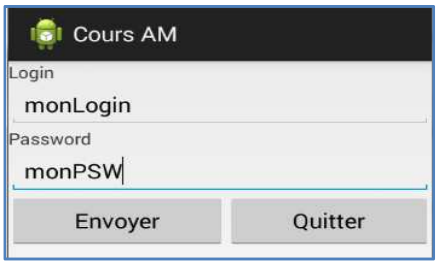
```
public class SecondActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_secondactivity);
    Button retour = (Button) findViewById(R.id.button1);

    retour.setOnClickListener(new OnClickListener() {
     @Override
       public void onClick(View v) {
           finish();
                    }
               });
       }
}
```

- Modifier le click du bouton ***Envoyer*** pour qu'il puisse lancer la Nouvelle Activité.

```
...
     envoyer.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
         Intent monIntent = new Intent(getApplicationContext(), SecondActivity.class);
         startActivity(monIntent);
                    }
             });
     . . .
  }
...
```

Exercice 3 :

Afficher le contenu des champs de texte *Login* et *Password* de l'activité principale dans la Nouvelle Activité. Vous pouvez utiliser deux (2) nouveaux composants *TextView* pour les afficher dans la Nouvelle Activité.



1- Dans l'activité principale (MainActivity.java)

```
  ...
public class MainActivity extends Activity {
  ...
public EditText userlogin;
public EditText userpsw;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
      ...
       userlogin = (EditText) findViewById(R.id.editText1);
         userpsw = (EditText) findViewById(R.id.editText2);

       envoyer.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
           Intent monIntent = new Intent(getApplicationContext(),
SecondActivity.class);
                  monIntent.putExtra("login", userlogin.getText().toString() );
                  monIntent.putExtra("psw", userpsw.getText().toString());
                  startActivity(monIntent);
                      }
              });
       ...
    }
  ...
```

2- Dans l'activité secondaire

2.1 ***Coté interface*** *(activity_secondactivity.xml)*

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/an
droid"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FFFF00"  >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="94dp"
        android:layout_marginTop="38dp" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="21dp" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="62dp"
        android:text="Retour" />

</RelativeLayout>
```

2.2 ***Coté code java*** (SecondActivity.java)

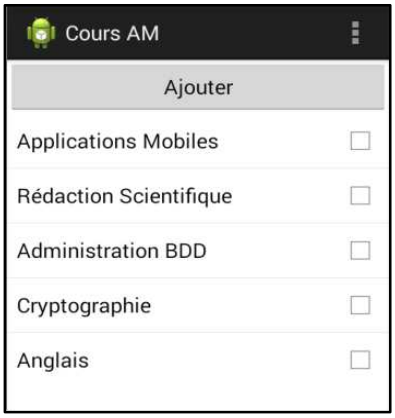
```
public class SecondActivity extends Activity {

@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_secondactivity);

      Button retour = (Button) findViewById(R.id.button1);
      TextView lg  = (TextView) findViewById(R.id.textView1);
      TextView psw = (TextView) findViewById(R.id.textView2);

      Intent intent = getIntent();
             lg.setText(intent.getStringExtra("login") );
            psw.setText(intent.getStringExtra("psw") );

      retour.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
            finish();
                  }
            });
      }
}
```

Solution TP4 (Liste, Dialogue, Menu)

Exercice 1 :

1- *Coté interface :* Le fichier layout : ***listModules.xml***

```
<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

<LinearLayout android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical" >

<Button  android:id="@+id/b1"
         android:layout_height="wrap_content"
         android:layout_width="match_parent"
         android:text ="Ajouter"/>

</LinearLayout>

<ListView android:id="@+id/list"
android:layout_height="wrap_content"
android:layout_width="match_parent"
android:choiceMode= "multipleChoice"    >
</ListView>
</LinearLayout>
```



2- *Coté code java (MainActivity.java)*

```
public class MainActivity extends Activity {

 public ArrayList<String> modules = new ArrayList<String>();
 public ArrayAdapter<String> adapter;
 public ListView mListView;

    @Override
   protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);

     modules.add("Applications Mobiles");
     modules.add("Rédaction Scientifique");
     modules.add("Administration BDD");
     modules.add("Cryptographie");
     modules.add("Anglais");


     setContentView(R.layout.listModules);

     Button b = (Button) findViewById(R.id.b1);

     b.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {

       final EditText et = new EditText(MainActivity.this);
                   et.setHint("Le nom du module");
```

```
AlertDialog.Builder dialog = new AlertDialog.Builder( MainActivity.this);
          dialog.setTitle("Ajouter un module");
          dialog.setView(et);

         dialog.setPositiveButton("OK", new DialogInterface.OnClickListener(){
                              public void onClick(DialogInterface dialog, int idBtn)
{
                                modules.add(et.getText().toString());
                                adapter.notifyDataSetChanged();
                                                    }
                                                });
         dialog.setNegativeButton("Annuler", null);


            AlertDialog confirm = dialog.create();
                     confirm.show();

                              }
               });


      mListView = (ListView) findViewById(R.id.list);

      adapter = new ArrayAdapter<String>
                     (MainActivity.this,
                      android.R.layout.simple_list_item_multiple_choice, modules);
      mListView.setAdapter(adapter);

   }
```

Exercice 2 :

1- Dans la même application, ajouter un menu qui contient 3 items (éléments) comme le montre **Fig.4**. L'item « ajouter » fait appel à la boite du dialogue qui permet d'insérer un nouveau module. On ajoute le menu « main.xml » dans res/menu.

```
<menu
xmlns:android="http://schemas.android.com/apk/res/a
ndroid"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.example.test.MainActivity" >

    <item
        android:id="@+id/add"
        android:showAsAction="ifRoom"
        android:title="@string/ajouter"/>
     <item
        android:id="@+id/upd"
        android:showAsAction="ifRoom"
        android:title="@string/modifier"/>

    <item
        android:id="@+id/del"
        android:showAsAction="ifRoom"
        android:title="@string/supprimer"/>

</menu>
```

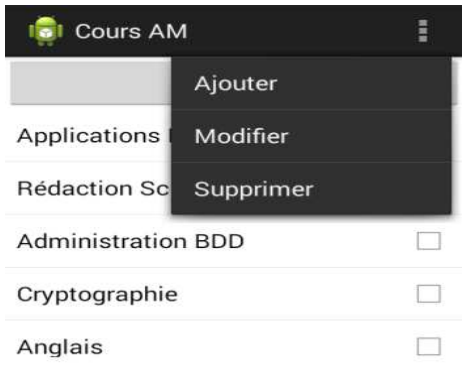


Fig.4

2- Modifier les attributs du menu pour qu'il s'affiche dans la barre de titre comme illustré dans **Fig.5**

2.1 Coté interface : dans res/menu/main.xml

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.example.test.MainActivity" >

    <item
        android:id="@+id/add"
        android:showAsAction="always"
        android:icon ="@drawable/ic_add"
        android:title="@string/ajouter"/>
     <item
        android:id="@+id/upd"
        android:showAsAction="always"
        android:icon ="@drawable/icupd"
        android:title="@string/modifier"/>

    <item
        android:id="@+id/del"
        android:showAsAction="always"
        android:icon ="@drawable/ic_del"
        android:title="@string/supprimer"/>

</menu>
```

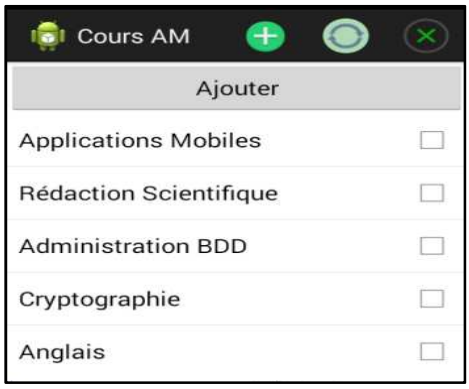


Fig.5

2.2 Coté code java *(MainActivity.java)* :

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

@Override
public boolean onOptionsItemSelected(MenuItem item) {
  int id = item.getItemId();

  if (id == R.id.add) {
     final EditText et = new EditText(MainActivity.this);
          et.setHint("Le nom du module");

     AlertDialog.Builder dialog = new AlertDialog.Builder(MainActivity.this);
     dialog.setTitle("Ajouter un module");
     dialog.setView(et);
     dialog.setPositiveButton("OK", new DialogInterface.OnClickListener(){
                                public void onClick(DialogInterface dialog,
int idBtn){
                                    modules.add(et.getText().toString());
                                adapter.notifyDataSetChanged();
                                    }
                                });
     dialog.setNegativeButton("Annuler", null);
     AlertDialog confirm = dialog.create();
     confirm.show();
        return true;
      }
     return super.onOptionsItemSelected(item);
    } . . .
```

3. Ajouter une action de click pour les éléments de la liste qui permet d'afficher, dans une boite de dialogue, l'indice de l'élément sélectionné.

```
. . .
@Override
protected void onCreate(Bundle savedInstanceState) {
  . . .
mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
 @Override
 public void onItemClick(AdapterView<?> parent, View view, int position, long
id) {

   String module = (String) mListView.getItemAtPosition(position);
   Toast.makeText(MainActivity.this,  "Le  module:"+module  +"  La  postion
est:"+position,
                Toast.LENGTH_LONG).show();
                       }
                 });
 . . .
}
```

Références

[1] Damien Guignard, Julien Chable, Emmanuel Robles, « Programmation Android De la conception au déploiement avec le SDK Google Android 2 », édition Eyrolles, 2010.

[2] Mark Murphy, « L'art du développement Android », édition PEARSON, 2009.

[3] Pierre Nerzic, Cours : « Programmation mobile avec Android » , IUT Lannion, 2016

[4] Frédéric Espiau (Apollidore), « Créez des applications pour Android », 2015

[5] Documentation officielle Android : https://developer.android.com/docs

[6] Sylvain HEBUTERNE, « Android Guide de développement d'applications Java pour Smartphones et Tablettes », Édition ENI, 2016

[7] Marc Dalmau, Cours : « Développement d'applications pour Android » , IUT de Bayonne-Pays Basque.

[8] Lars Vogel, « Tutoriel sur l'utilisation de base de données SQLite sous Android », 2013