MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOHAMED KHIDER DE BISKRA
FACULTE DES SCIENCES EXACTES ET SCIENCES DE LA NATURE ET DE LA VIE
DEPARTEMENT D'INFORMATIQUE

Cours Applications Mobiles

3ème Année Licence LMD (semestre 6)

Préparé par : Dr. Somia SAHRAOUI

Année universitaire 2019-2020

Table des matières

✓ Informations sur la matière

- Crédits : 5
- Coefficient : 3
- Mode d'évaluation : 60% Examen + 40 % évaluation continue.

✓ Objectif général du cours

L'objectif visé est que chaque étudiant puisse concevoir et développer des applications Android simples.

✓ Connaissances pré-requises

Pour atteindre l'objectif principale de l'enseignement, il est primordiale d'avoir des connaissances préalables sur : les principes des systèmes d'exploitation ordinaires, Java, XML, SQL.

CHAPITRE 1

Les Applications Mobiles : Généralités et Contextualisation

Objectifs unitaires de l'enseignement :

- Avoir une idée générale sur les systèmes d'exploitation pour équipements mobiles,
- Connaitre les différentes stratégies de développement de tels systèmes d'exploitation,
- Connaître les trois types d'applications mobiles.

1. Introduction

De nos jours, les équipements mobiles (smartphones, les tablettes électroniques, montres intelligentes, etc.) occupent une zone capitale de notre vie quotidienne. En effet, l'attachement aux dispositifs mobiles est renforcé par les différentes applications mobiles que nous téléchargeons pour combler nos besoins multiples.

Les applications mobiles ont diverses missions ; on trouve des applications crées pour supporter les communications au sens large, des applications de sport, d'apprentissage, de tourisme, etc. dont l'avantage commun entre les applications mobiles est d'améliorer la qualité de vie et d'expérience de leurs utilisateurs.

A travers ce module, les étudiants de la 3ème année licence Informatique vont apprendre les fondements du développement des applications mobiles, particulièrement les applications mobiles tournant sur une plateforme Android.

2. Aperçu général sur les systèmes d'exploitation pour équipements mobiles

Le monde des systèmes d'exploitation mobiles a connu une révolution énorme. Contrairement aux OS classiques qui étaient destinés aux ordinateurs, les OS pour équipements mobiles doivent être adaptés aux contraintes des dispositifs mobiles à lesquels ils sont destinés, particulièrement la contrainte énergétique, la limitation des ressources de stockage mémoire, ainsi que de traitement.

De nombreux systèmes d'exploitation pour dispositifs de communication mobiles se sont émergés. Une féroce concurrence naissait entre les différents fondateurs dont l'objectif commun étant d'attirer un maximum possible de clients.

Les principaux facteurs de succès sont globalement représentés par :

- L'efficacité du système : ce point comprend davantage de critères tels que la flexibilité de la gestion des données et des ressources, sécurité, interface graphique adaptée, ..
- La considération de la QoE (*Quality of Experience*) de la clientèle : respecter les préférences et les besoins des utilisateurs sur différents plans : esthétique, économie des ressources énergétiques, etc.

Néanmoins, le facteur qui décide de la popularité du système c'est bien le bon compromis entre l'efficacité et le coût.

2.1. Le système Android

Développé par Google. Il est parmi les systèmes d'exploitation mobiles les plus célèbres. Il est actuellement le système le plus adopté à travers le monde. Cette célébrité

est la conséquence de ses multiples avantages, dont principalement la disponibilité du code (*open source*), la bonne considération des exigences des clients, ainsi que la réalisation d'un rapport très satisfaisant entre l'efficacité et coût.

2.2. Le système iOS

Développé au sein de la société Apple. Il est un autre système d'exploitation sur le marché des équipements de télécommunication mobile. Il se considère comme un concurrent important du système Android, mais comme les coûts avec ce système est nettement onéreux, cela a sensiblement affecté son large adoption.

On trouve également d'autres systèmes qui sont beaucoup moins célèbres et/ou moins efficaces comme le système Windows Phone, de Microsoft, BADA de Samsung, etc.

En fait, les deux systèmes d'exploitation Android et iOS appartiennent à deux stratégies de développement / commercialisation différentes :

- Un standard pour différents types d'équipements

Suivant cette stratégie un système d'exploitation suffisamment efficace est développé par un large groupe d'experts dans le domaine du software. Le système est destiné à être un système standard pour l'exploitation des dispositifs mobiles provenant de différents fabricants. Telle stratégie est adoptée par Google et le système Android par différents types de dispositifs, tels que les dispositifs Samsung, Oppo, Hwawei, etc. Telle stratégie présente les avantages et les inconvénients suivants :

- **Avantages**
  - Les systèmes développés suivant cette approche sont généralement plus robustes car c'est un groupe élargi d'experts qui les élaborent.
  - Interopérabilité des applications est un avantage majeur.
  - les systèmes appartenant à cette famille sont caractérisés par l'extensibilité et une extrême facilité de développement car ils sont généralement open source.

- **Inconvénients**
  - Toute anomalie liée aux performances et/ou à la sécurité va affecter l'ensemble de tous les dispositifs adoptant le système
  - Risque d'exploration non optimale des ressources matériel des différents équipements.

- **Un système d'exploitation spécifique pour chaque type de dispositif**

  Contrairement à la politique précédente, certains constructeurs des dispositifs de communication mobiles préfèrent avoir des systèmes d'exploitation spécialement développés pour ce qu'ils produisent comme matériel. Cela dans l'objectif d'assurer de l'autonomie vis-à-vis une partie centrale qui développe des systèmes standards. Le système iOS d'Apple est l'exemple le plus réussi des systèmes qui naissaient sous cette stratégie. D'autres exemples de systèmes existent mais qui demeurent méconnus comme le système BADA de Samsung.

Les avantages et les inconvénients de cette approche sont énumérés ci-dessous.

- **Avantages** :
  - La sécurité est meilleure avec les systèmes de cette politique
  - Exploration optimale et adaptée au matériel.
  - Autonomie : éviter les risques de retrait des droits d'utilisation du système. A ce niveau, on rappelle le problème qui est survenu entre Hwawei et Google.
- **Inconvénients :**
  - Mauvaise interopérabilité; différentes applications pour différentes plateformes, chose qui rend délicat le partage de données et l'interaction entre les applications fonctionnant sur des plateformes (système d'exploitation et support matériel) hétérogènes.

Le tableau ci-dessous compare les deux systèmes Android et iOS :

Table 1.1 Comparaison entre les fameux systèmes Android et iOS.

Critère	Système Android	Système iOS
Fondateur	Google	Apple
Accessibilité de la licence	Licence libre	Non
Langage de développement	Principalement Java	Objective C
Sécurité	Exposé aux menaces de sécurité	Bonne sécurité
Services Cloud	Intégration native avec Google drive	Intégration native avec iCloud
Coût de développement	Faible	Elevé et trop restreint
Coût des équipements adoptant le système	Abordable	Elevé

3. Les types d'applications mobiles

On distingue trois types d'applications mobiles : les applications natives, les applications web et les applications hybrides.

3.1. Les applications natives

Bien que le terme ''application mobile native'' soit méconnu du grand public, cela représente le type dominant des applications mobiles que nous téléchargeons et utilisons quotidiennement. Une application native c'est une application qui est développée spécifiquement pour un système d'exploitation. Par conséquent, l'application native est destinée à fonctionner seulement sur les dispositifs qui supportent le système d'exploitation pour lequel l'application a été développée. Donc, une application pour iOS ne fonctionnerait pas sur Android et vice-versa. Bien évidemment, cela signifie que le langage de programmation est différent d'un système d'exploitation à un autre. Par exemple, iOS utilise le langage Objective C, tandis qu'Android utilise Java. C'est pour cela que les développeurs précisent sur quelle plateforme ils développent. Outre, un développeur Android n'est pas automatiquement développer sur iOS et vice-versa.

- **Les avantages du développement natif**
  - L'accès à l'application native est rapide et facile.
  - Elle permet un accès plus facile à toutes les fonctionnalités du téléphone (par exemple la caméra, l'accéléromètre ou même le micro).
  - Ne nécessite pas de connexion internet pour être utilisée.
  - Elle est plus fiable et se retrouve en bonne sécurité (conséquence du non nécessité de connexion internet).

- **Les inconvénients du développement natif**
  - Le développement des applications mobiles natives est assez coûteux car il requière beaucoup du temps et une bonne maitrise des outils de développement sur la plateforme ciblée.

3.2. Les applications web

Il s'agit cette fois-ci d'une application qui fonctionne comme un site web. La version mobile d'un site web, c'est une application web. Elle est dite multiplateforme car elle ne dépend pas du tout de la plateforme, de système d'exploitation ni même du code. Cela implique que les utilisateurs n'auront pas à installer l'application sur leur smartphones. Ce qui présente un très point fort majeur.

Le but principal d'une application web est de rendre du contenu opérationnel et disponible sur dispositif mobile.

❖ **Les avantages de l'application web**

- Portabilité (application multiplateforme).
- Une application web coûte moins cher qu'une application native.
- Une application web ne surcharge pas les ressources mémoire des dispositifs mobiles (car pas de besoin d'installation préalable).

❖ **Les inconvénients de l'application web**

- L'application web nécessite de la connexion internet.
- Risque de sécurité sont élevés avec les applications web.

3.3. Les applications hybrides

On peut trouver des applications mobiles hybrides entre les applications natives et les applications web. Une application hybride comporte des parties natives et des parties de la vue sont des affichages web. Une application hybride allie les avantages des deux types d'applications ; elle est par exemple plus rapide qu'une application web, et au même temps, moins chère à développer par rapport à une application native.

Les types d'applications mobiles auxquels nous allons nous intéresser dans ce cours, ce sont bien les applications natives (principalement) et les applications hybrides.

CHAPITRE 2

La Plateforme Android

Objectifs unitaires de l'enseignement :

- Découvrir l'architecture et les composants de la plateforme Android
- Avoir une idée sur le SDK Android.

1. Présentation du système Android

Dans cette partie, nous allons nous focaliser sur la présentation de l'architecture du système Android ainsi que ses principales caractéristiques.

1.1. Architecture du système Android

L'architecture du système Android est constituée de 4 couches et 5 composants comme le montre la figure suivante.

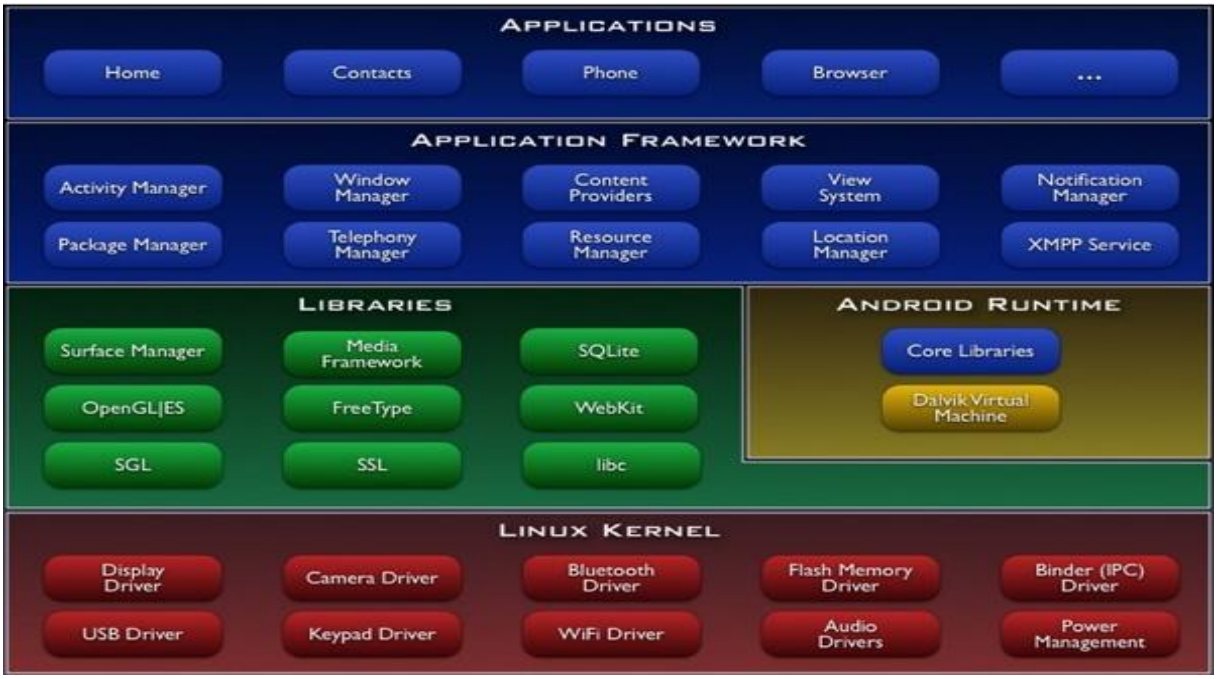


Figure 2.1. Modèle en couches du système Android.

1.1.1. La couche Linux Kernel

Android est basé sur un noyau linux réduit qui implémente les programmes pilotes des différents modules et parties du téléphone mobile.

Positionné au bas de la pile logicielle Android, le noyau Linux fournit un niveau d'abstraction entre le matériel de l'appareil et les couches supérieures de la pile logicielle Android. Basé sur Linux version 2.6, le noyau fournit des services système multitâches préemptifs de bas niveau tels que la gestion de la mémoire, des processus et de l'énergie en plus de fournir une pile réseau et des pilotes de périphériques pour le matériel tel que l'affichage de l'appareil, le Wi-Fi et l'audio.

Il est toutefois important de noter qu'Android utilise uniquement le noyau Linux. Cela dit, il convient de noter que le noyau Linux a été initialement développé pour être utilisé dans les ordinateurs traditionnels sous la forme de bureaux et de serveurs. En fait, Linux est désormais le plus largement déployé dans les environnements de serveurs

d'entreprise critiques. C'est un témoignage à la fois de la puissance des appareils mobiles d'aujourd'hui et de l'efficacité et des performances du noyau Linux que nous trouvons ce logiciel au cœur de la pile de logiciels Android.

1.1.2. Les librairies

En plus d'un ensemble de bibliothèques de développement Java standard (prenant en charge des tâches générales telles que la gestion des chaînes, la mise en réseau et la manipulation de fichiers), l'environnement de développement Android comprend également les bibliothèques Android. Il s'agit d'un ensemble de bibliothèques basées sur Java qui sont spécifiques au développement Android. Des exemples de bibliothèques de cette catégorie incluent les bibliothèques du cadre d'application en plus de celles qui facilitent la construction de l'interface utilisateur, le dessin graphique et l'accès à la base de données.

Voici un résumé de certaines des principales bibliothèques Android disponibles pour le développeur Android:

- **android.app :** Fournit un accès au modèle d'application et est la pierre angulaire de toutes les applications Android.
- **android.content :** Facilite l'accès au contenu, la publication et la messagerie entre les applications et les composants d'application.
- **android.database :** Utilisé pour accéder aux données publiées par les fournisseurs de contenu et inclut des classes de gestion de base de données SQLite.
- **android.graphics :** Une API de dessin graphique 2D de bas niveau comprenant des couleurs, des points, des filtres, des rectangles et des toiles.
- **android.hardware :** Présente une API permettant d'accéder au matériel tel que l'accéléromètre et le capteur de lumière.
- **android.opengl :** Une interface Java vers l'API de rendu graphique 3D OpenGL ES.
- **android.os :** Fournit aux applications l'accès aux services standard du système d'exploitation, y compris les messages, les services système et la communication interprocessus.
- **android.media :** Fournit des classes pour permettre la lecture audio et vidéo.
- **android.net :** Un ensemble d'API permettant d'accéder à la pile réseau. Inclut android.net.wifi, qui donne accès à la pile sans fil de l'appareil.
- **android.print :** Comprend un ensemble de classes qui permettent d'envoyer du contenu aux imprimantes configurées à partir des applications Android.
- **android.provider :** Un ensemble de classes de commodité qui donnent accès aux bases de données standard du fournisseur de contenu Android telles que celles gérées par les applications de calendrier et de contact.

- **android.text :** Utilisé pour afficher et manipuler du texte sur l'écran d'un appareil.
- **android.util :** Un ensemble de classes utilitaires pour effectuer des tâches telles que la conversion de chaînes et de nombres, la gestion XML et la manipulation de la date et de l'heure.
- **android.view :** Les blocs de construction fondamentaux des interfaces utilisateur des applications.
- **android.widget :** Une riche collection de composants d'interface utilisateur prédéfinis tels que des boutons, des étiquettes, des vues de liste, des gestionnaires de disposition, des boutons radio, etc.
- **android.webkit :** Un ensemble de classes destiné à permettre aux capacités de navigation Web d'être intégrées dans les applications.

En plus des bibliothèques basées sur Java dans le *Runtime* Android, cette couche contient également des bibliothèques basées sur C / C ++. Les bibliothèques C / C ++ sont incluses pour remplir un large éventail de fonctions, y compris le dessin graphique 2D et 3D, la communication SSL (Secure Sockets Layer), la gestion de la base de données SQLite, la lecture audio et vidéo, etc.

Dans la pratique, le développeur d'application Android typique accède à ces bibliothèques uniquement via les API de bibliothèque de base Android basées sur Java. Dans le cas où un accès direct à ces bibliothèques est nécessaire, cela peut être réalisé en utilisant le Kit de développement natif Android (NDK), dont le but est d'appeler les méthodes natives des langages de programmation non Java ou Kotlin (tels que C et C ++) à partir du code Java à l'aide de Java Native Interface (JNI).

1.1.3. L’environnement d’exécution (Runtime) Android

Les applications Android sont essentiellement développées en Java mais, elles ne sont pas supportées par la JVM java car il doit y avoir une machine virtuelle averties des diverses contraintes du système Android qui tourne sur des dispositifs limités en ressources de stockage mémoire, de traitement et même d’énergie.

’’Dalvik’’ est le nom de la machine virtuelle open-source utilisée sur le système Android. Cette machine virtuelle exécute des fichiers ’’.dx’’, plus ramassés que les fichiers .class classiques. Ce format évite par exemple la duplication des String constantes. La machine virtuelle utilise elle-même moins d'espace mémoire et l'adressage des constantes se fait par un pointeur de 32 bits.

La version java qui permet le développement mobile sous Android est plus réduite ; les fonctionnalités qui ne sont pas nécessaires pour le développement mobile sont désormais exclues.

La figure ci-dessous montre les étapes de compilation et d'exécution des programmes java sous Android :

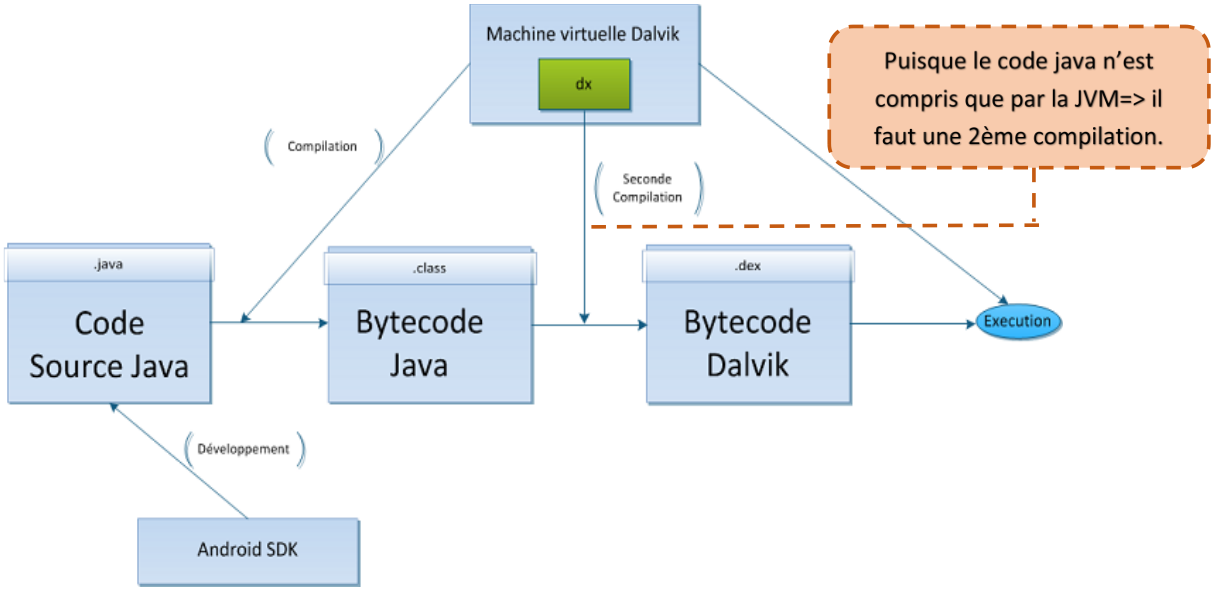


Figure 2.2. Etapes de compilation des programmes java sous Android.

1.1.4. Application Framework

Cette couche est un ensemble de services qui forment collectivement l'environnement dans lequel les applications Android s'exécutent et sont gérées. Elle met en œuvre le concept selon lequel les applications Android sont construites à partir de composants réutilisables, interchangeables et remplaçables. Ce concept va encore plus loin dans la mesure où une application est également en mesure de publier ses capacités ainsi que toutes les données correspondantes afin qu'elles puissent être trouvées et réutilisées par d'autres applications.

La couche Application Framework comprend les services-clés suivants :

- **Gestionnaire d'activités :** Contrôle tous les aspects du cycle de vie des applications et de la pile d'activités.
- **Fournisseurs de contenu :** Permet aux applications de publier et de partager des données avec d'autres applications.
- **Gestionnaire de ressources** : Donne accès à des ressources intégrées non codées telles que des chaînes, des paramètres de couleur et des dispositions d'interface utilisateur.
- **Gestionnaire de notifications** : Permet aux applications d'afficher des alertes et des notifications à l'utilisateur.

- **Système d'affichage :** Un ensemble extensible de vues utilisé pour créer des interfaces utilisateur d'application.
- **Gestionnaire de packages** : Le système par lequel les applications peuvent trouver des informations sur d'autres applications actuellement installées sur l'appareil.
- **Gestionnaire de téléphonie :** Fournit à l'application des informations sur les services de téléphonie disponibles sur l'appareil, telles que l'état et les informations sur l'abonné.
- **Gestionnaire d'emplacement :** Donne accès aux services de localisation permettant à une application de recevoir des mises à jour sur les changements d'emplacement.

1.1.5. Les Applications

Les applications sont situées en haut de la pile logicielle Android. Ceux-ci comprennent à la fois les applications natives fournies avec l'implémentation Android particulière (par exemple, les applications de navigateur Web et de messagerie) et les applications tierces installées par l'utilisateur après l'achat de l'appareil.

2. Structure du Projet de développement Android

Tout projet de développement d'application Android est organisé en trois dossiers, comme le montre la figure :

- Le dossier Manifests : ce dossier contient le fichier ''AndroidManifest.xml'' qui indispensable pour chaque application. Ce fichier déclare les permissions nécessaires pour l'application (comme l'utilisation de la connexion internet, l'accès au répertoire téléphonique, etc.). Voir **Annexe 2** pour une vision plus élargie sur l'ensemble des permissions possibles.

Le fichier ''AndroidManifest.xml'' permet également de déclarer les composants de l'application mobile.

La structure générale du fichier est comme suit :

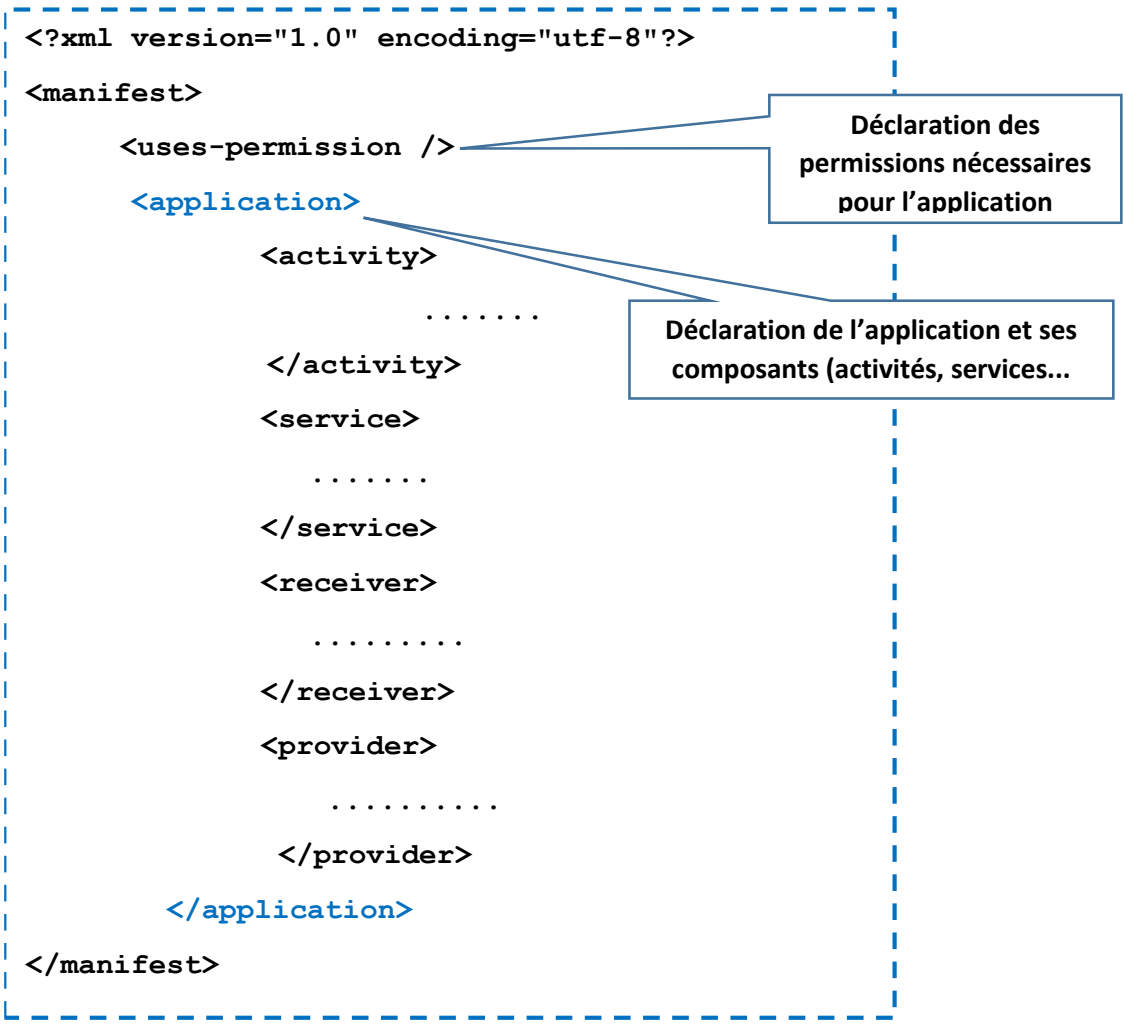


Figure 2.3. Structure générale du fichier AndroidManifest.xml

- Le dossier Java : le dossier Java contient le code source java de l'application.

- Le dossier Res : contient les ressources xml nécessaires pour l'application comme l'interface graphique (fichier layout.xml). Nous reviendrons sur les détails des ressources de l'application mobile Android (leur création, manipulation, etc.) dans le chapitre suivant.

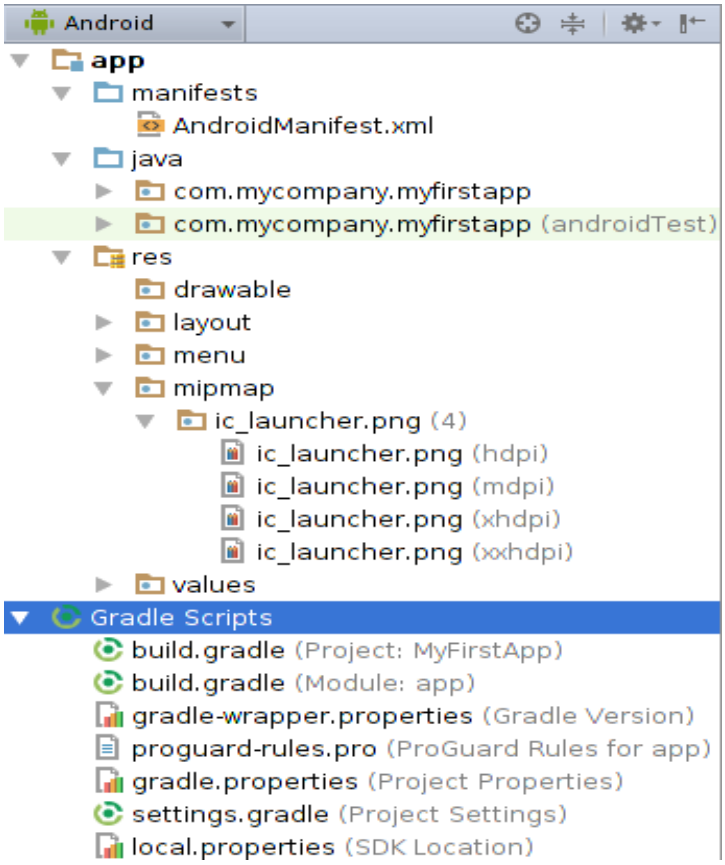


Figure 2.4. Structure du projet de développement Android

3. Aperçu général sur le SDK Android

Le SDK Android (*Software Developement Kit*) est un ensemble complet d'outils de développement1. Il inclut un débogueur, des bibliothèques logicielles, un émulateur, de la documentation, des exemples de code et des tutoriels. A chaque fois que Google publie une nouvelle version d'Android, un SDK correspondant est également publié. Pour pouvoir écrire des programmes avec les dernières fonctionnalités, les développeurs doivent télécharger et installer le SDK de chaque version pour le téléphone en question.

Les développeurs peuvent utiliser n'importe quel éditeur de texte pour modifier les fichiers Java et XML, puis utiliser les outils en ligne de commande (Java Development Kit et Apache Ant sont obligatoires) pour créer, construire et déboguer les applications Android ainsi que contrôler des périphériques Android (pour déclencher un redémarrage, installer un logiciel à distance ou autre). L'IDE officiellement supporté était Eclipse combiné au module d'extension avec les outils de développement d'Android (ADT) mais depuis 2015, Google avait lancé Android Studio, qui devient alors l'IDE officiel pour ce kit de développement d'Android.

Les étapes d'installation et de configuration de l'outil Android Studio sont expliquées dans **Annexe 1**.

Les plates-formes de développement compatibles avec le SDK incluent les systèmes d'exploitation Windows, Linux (toute distribution Linux récente) et Mac OS X (10.4.9 ou version ultérieure). De plus, les composants du SDK Android peuvent être téléchargés séparément.

Le SDK Android comprend ainsi deux éléments essentiels :

- **Un émulateur :** Le SDK comprend un émulateur qui permet de simuler les différentes versions d'Android et tester les applications développées sur différents types de dispositifs mobiles émulés tels que : le smartphones, smart TV, montre intelligente, etc.

- **Android Debug Bridge** : L'Android Debug Bridge (ADB) est un outil inclus dans le package Android SDK. Il est composé d'un programme client et d'un programme serveur qui communiquent entre eux. L'ADB permet de faire les opérations suivantes :

  - Copier un fichier avec la commande : `adb push source destination` ;

  - Accès à la console Android : `adb shell`.

  - Sauvegarde de la mémoire ROM : `adb backup –all`.

  - Installation du logiciel : `adb install NomDuFichierApk`.

CHAPITRE 3

Les activités et les ressources de l'application mobile Android

Objectifs unitaires de l'enseignement :

- Connaître ce qui est le composant activité de l'application mobile Android.
- Apprendre à définir et manipuler les activités d'une application
- Connaître les ressources de l'application et comment les exploiter.

1. Introduction

Systématiquement, une Application Android est composée de quatre éléments essentiels :

- Les activités (android.app.Activity): il s'agit d'une partie de l'application présentant une vue graphique à l'utilisateur
- Les récepteurs d'Intents (android.content.BroadcastReceiver): composant permettant de déclarer que l'application est capable de recevoir et répondre à des Intents.
- Les fournisseurs de contenus (android.content.ContentProvider): pour le partage d'informations au sein ou entre applications mobiles.
- Les services (android.app.Service): il s'agit du composant de l'application qui exécute une tâche de fond sans présenter une vue graphique à l'‘utilisateur.

Nous abordons, à partir de ce chapitre, les composants de l'application Android un par un, ainsi que les éléments se rapportant avec chaque élément. Dans ce qui suit, nous commençant par les activités.

2. Les activités

L'activité est un élément important dans les applications mobiles Android. Il s'agit d'une vue graphique (un écran) présentant une interface graphique à l'utilisateur. Autrement dit, l'activité est un conteneur de l'interface graphique que l'application présente à l'utilisateur. A titre d'exemple, une application e-mail a une activité principale qui présente à l'utilisateur une liste des mails reçus. Pour composer un mail et l'envoyer à un destinataire bien déterminé, l'application présente une autre vue graphique (activité).

- **Remarque** : Une application mobile doit avoir au moins une activité. Cela veut dire que l'activité est un composant obligatoire dans les applications mobiles. La vue graphique qui apparait en premier, au démarrage de l'application, est appelée Activité principale (*main activity*).

2.1. Le cycle de vie de l'activité

Les activités ont un cycle de vie tout comme les processus. En effet, il y a cinq états par lesquels l'activité peut passer. Entre chaque état et état l'activité exécute des méthodes dites callback. Dans les méthodes callback du cycle de vie, on peut déclarer le comportement de l'activité lorsque l'utilisateur quitte et réintègre l'activité.

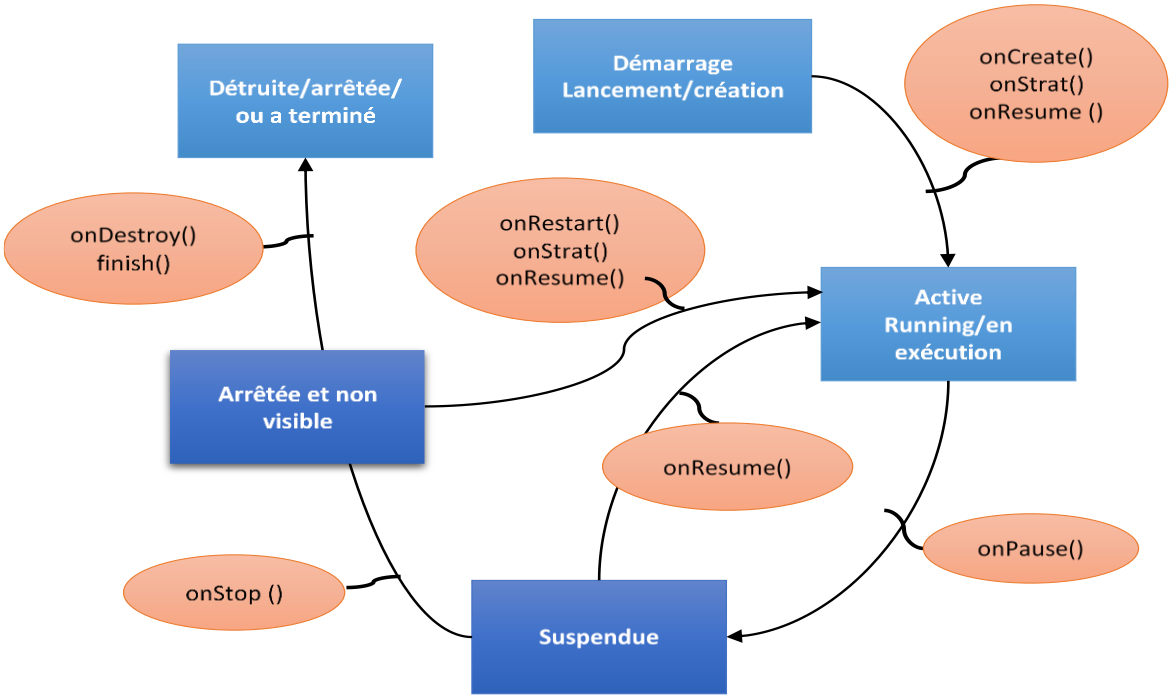


Figure 3.1. Le cycle de vie des activités.

Explication du rôle de chaque méthode callback du cycle de vie de l'activité :

- **onCreate () :** la méthode onCreate () est évoquée pour gérer les actions à entreprendre quand l'activité est créée. A ce niveau-là, l'activité n'est pas encore affichée.
- **onStart () :** la méthode callback onStart () est évoquée pour gérer ce qui doit se faire à l'affichage de l'activité.
- **onResume () :** cette méthode est appelée quand l'utilisateur commence à interagir avec l'activité.
- **onPause () :** appelée quand l'activité est visible mais pas d'interaction avec l'utilisateur. Cela survient quand une interruption intervient, exemple : appel téléphonique. Dans ce cas l'activité est partiellement visible et elle perd l'appui de l'utilisateur.
- **onStop () :** appelée quand l'activité n'est plus visible, mais elle est maintenue dans la mémoire et l'utilisateur a toujours la possibilité d'y revenir. Ce cas arrive quand par exemple l'utilisateur appui sur la touche home.
- **onRestart () :** est évoquée quand l'utilisateur tente de mettre en premier plan l'activité qui a été arrêtée.
- **onDestroy () :** la méthode onDestroy () est appelée quand l'activité termine la mission qui lui a été confiée.si on reprend l'exemple de l'application mail, l'activité destinée à la rédaction d'un email disparaît une fois l'utilisateur envoie le mail. La méthode onDestroy () peut être évoquée quand l'activité doit être

détruite par Android afin de récupérer de l'espace mémoire pour l'attribuer à une tache plus prioritaire.

2.1.1. Création de l'activité dans le code Java

Pour définir une activité de l'application, il faut déclarer une classe qui hérite de la classe Activity et qui surcharge les méthodes callback du cycle de vie de l'acticité. La redéfinition de la méthode onCreate () est obligatoire. Cependant, la redéfinition du reste des méthodes callback ne l'en est pas. De plus, à l'intérieur de chaque méthode callback redéfinie, il faut évoquer la méthode callback correspondante dans la classe mère pour que la machine virtuelle Dalvik exécute en plus, le code existant dans la méthode callback de la classe mère. Si on oublie de faire cet appel, seul le code de la méthode redéfinie qui sera exécuté.

Pour créer une activité en Java, il suffit de suivre le modèle ci-dessous :

```
==========================================================
public class nom_activite extends Activity
{
      @Override
      Protected void onCreate (Bundle SavedInstanceState)
      {     super.onCreate (SavedInstanceState);
            ....
      }
      @Override
      Protected void onPause ()
      {     super.onPause ();
            ....
      }
      @Override
      Protected void onStop ()
      {     super.onStop ();
            ....
      }
      @Override
      Protected void onDestroy ()
      {     super.onDestroy ();
```

```
            ....
        }
}
```

==========

- **Remarques**

- L'objet `Bundle` passé en paramètre de la méthode onCreate() permet de sauvegarder et restaurer les valeurs des objets de l'interface de l'activité qui a été déchargée de la mémoire. Par exemple : Si l'utilisateur appuie sur la touche ''Home'', Android peut décharger les activités de l'application de la mémoire. Et au cas où l'utilisateur revient sur l'application, i peu se trouver que l'application perde les valeurs saisies dans les zones de texte par exemple. A cet effet, il est important de sauvegarder l'état des activités de l'application de l'objet Bundle communément appelé `SavedInstanceState`.
- Il est vivement recommandé d'éviter de mettre trop de code dans la méthode `onCreate`() pour éviter le risque de blocage/ralentissement du démarrage de l'activité et l'application toute entière.

2.1.2. Définition de l'activité dans le fichier AndroidManifest.xml

La déclaration des activités de l'application dans le fichier AndroidManifest.xml de la même application est obligatoire. Cela se fait à l'aide de la balise **<activity ...> </activity>** et on utilise autant de balises <activity> qu'il ait d'activités dans l' application à développer. Avec certains outils de développement Android, tel que Android Studio, la génération des parties correspondantes aux activités de l'application se fait de façon automatique dans le fichier Manifest.

Ci-dessous, un modèle pour la déclaration de l'activité dans le fichier Manifest de l'application.

==========

```
<? xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    .......>

    <application

        <activity
             android:name=".nom_Activite"
            android:label="nom_app" >
```

```
                ……
        </activity>

    </application>
</manifest>
============================================================
```

- **Exemple :**

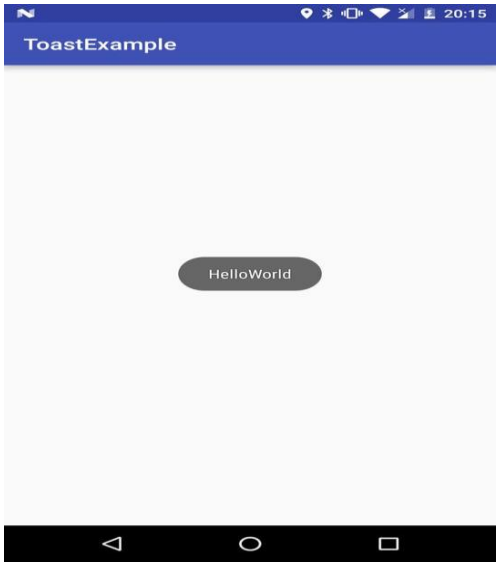
Exemple d'une application ayant une seule activité. Cette dernière affiche le message Hello dans un Toast.

```
public class HelloAct extends Activity
{
      @Override
      Protected void onCreate (Bundle SavedInstanceState)
      { Super.onCreate (SavedInstanceState);
      Toast.makeText(this, "HelloWorld", Toast.LENGTH_LONG) .show();
      }
}
```

Et dans le fichier AndroidManifest.xml, on devrait avoir :

```
<? xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.exampleandroidapps.example1"
    android:versionCode="1"
    android:versionName="1.0" >

    <application

        <activity

             android:name=".HelloAct"
            android:label="@string/ToastExample" >

        </activity>

    </application>
</manifest>
```

Le résultat graphique est :



3. Les ressources

En plus du code, l'application Android est composée également des ressources. Les ressources sont utilisées pour tout, de la définition des couleurs, des images, du son, des menus, des chaînes de caractères, etc. Tout est défini dans ces fichiers de ressources et peut ensuite être référencé dans le code de l'application. Tout projet Android a un dossier nommé ''res/'' contenant les ressources de l'application.

Les ressources de l'application peuvent être manipulées dans le code au travers de la classe statique ''R'' et elles sont accessibles dès qu'elles soient déclarées dans les fichiers xml. Il est important de signaler que R est régénéré automatiquement à chaque changement dans le code.

Le tableau ci-dessous illustre quelques types de ressources, les fichiers ans lesquels elles doivent être déclarées et la façon dont on les accède avec R.

Table 3.1. Exemples de ressources avec leur définition et chemin d'accès.

Type	Fichier	Accès
Couleurs	res/values/colors.xml	**R.color.nom_couleur**
Images	res/drawable.xml	**R.drawable.nom_img**
Strings	res/values/strings.xml	**R.string.non_chaine**
Interface	res/layout/nomInterface.xml	**R.layout.nomInterface**
Style	res/values/nomFichier.xml	**R.style.nomStyle**

3.1. Définition des ressources

Dans ce qui suit, on donne quelques exemples de définition des ressources de l'application Android.

3.1.1. Définition des ressources chaines de caractère

Les chaines de caractères de l'application on les définit dans le fichier ''strings.xml'' qui se trouve dans le chemin ''res/values/strings.xml'', suivant la syntaxe ci-dessous.

```
============================================================
<?xml version="1.0" encoding="utf-8"?>

<resources>

    <string name="NomRessource">Valeur_Ressource</string>

</resources>

============================================================
```

- **Exemple :** L'exemple ci-dessous montre comment définir les ressources de type chaine. Deux chaines ont été créées : la chaine AppName pour le nom de l'application et la chaine link pour sauvegarder le lien `https://developer.android.com`.

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
```

```
    <string name="AppName">Hello!</string>

    <string name="label">Test</string>

    <string name="link">https://developer.android.com</string>

</resources>
```

3.1.2. Définition d'une ressource couleur

Dans le fichier ‘’colors.xml’’ qui se retrouve dans le chemin : ‘’res/values/colors.xml’’. Il est à noter que les couleurs sont identifiées par des valeurs hexadécimales dans Android.

Syntaxe de définition :

```
============================================================

<?xml version="1.0" encoding="utf-8"?>

<resources>

    <color name="Nom_Ressource">valeurHexa_Correspondante</color>

</resources>

============================================================
```

- **Exemple :**

```
<?xml version="1.0" encoding="utf-8"?>

<resources>

    <color name="red">#F00</color>

    <color name="white">#FFFFFF</color>

    <color name="yellow">#FFFF00</color>

    <color name="fuchsia">#FF00FF</color>

</resources>
```

3.1.3. Définition d'une ressource image

On définit la ressource image dans un fichier dont l'extension ‘’.png’’. Le fichier se retrouve dans le chemin : ’’res/drawable/ nom_fichierIMG.png’’.

```
============================================================
```

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@[package:]drawable/drawable_resource"
    android:antialias=["true" | "false"]
    android:dither=["true" | "false"]
    android:filter=["true" | "false"]
    android:gravity=["top" | "bottom" | "left" | "right" |
           "center_vertical" |"fill_vertical" | "center_horizontal"
           |"fill_horizontal" | "center" | "fill" | "clip_vertical"
           | "clip_horizontal"]
    android:tileMode=["disabled" | "clamp" | "repeat" | "mirror"] />
```

==========

Où :

- **<bitmap>** Définit la source bitmap et ses propriétés.
- **xmlns:android** Définit l'espace de noms XML, qui doit être : "http://schemas.android.com/apk/res/android". Ceci n'est requis que si le `<bitmap>` est l'élément racine.
- **android:src** Paramètre obligatoire qui fait référence à la source de l'image.
- **android:antialias** Booléen pour activer ou désactiver l'anticrénelage.
- **android:tramage** Booléen pour activer ou désactiver le tramage du bitmap si le bitmap n'a pas la même configuration en pixels que l'écran (par exemple: un bitmap ARGB 8888 avec un écran RGB 565).
- **android:filtre** Booléen pour activer ou désactiver le filtrage bitmap. Le filtrage est utilisé lorsque le bitmap est réduit ou étiré pour lisser son apparence.
- **android:gravité** Définit la gravité du bitmap. La gravité indique où positionner l'image dans son conteneur.
- **android:tileMode** Définit le mode de tuile. Lorsque le mode est activé, le bitmap est répété.

❖ **Exemple**

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/MyImage"
    android:tileMode="repeat" />
```

3.1.4. Définition de la ressource style

Une ressource ‘’style’’ présente un format pour plusieurs éléments (taille de texte, la couleur, ..) d’une vue graphique ou l’activité toute entière. La ressource style est définie dans un fichier XML dans le chemin : ‘’res/values/nom-fichier.xml’’.

Syntaxe de définition :

==========

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style
        name="nom_style"
        parent="@style/style_à_heriter">
        <item name="nom_item"> valeur_style </item>
    </style>
</resources>
```

==========

- **Exemple**

L’exemple ci-dessous présente un style définissant la taille et la couleur du texte.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="exempleStyleText" parent="@style/Text">
        <item name="android:textSize">10sp</item>
        <item name="android:textColor">#008</item>
    </style>
</resources>
```

3.2. Référencer les ressources

Pour référencer les ressources et les utiliser dans d'autres fichiers XML par exemple un fichier de l'interface graphique, il faut respecter la notation suivante :

```
@type_ressource/nom_ressource
```

❖ **Exemple :** Si on veut que le texte qui s'affiche dans l'objet graphique Bouton soit de couleur rouge. Avec l'image MyImage comme image du fond.

```
<Button
   ……
  android:text="@string/label"
 anroid:style="@style/exempleStyleText
  android:background="@drawable/MyImage"
/>
```

- **Remarque :** Nous détaillons la partie l'interface graphique dans le chapitre suivant.

3.3. Récupération et manipulation des ressources

Dans certains cas, on souhaite récupérer dynamiquement des ressources dans le code Java de l'application en utilisant uniquement le nom de la ressource. Cela peut être réalisé en utilisant la méthode `getResources`() dans une activité de la façon suivante :

```
getResources().getType_resource_method(R.type.id_resource);
```

❖ **Exemple**

```
String s = getResources().getString(R.string.label) ;
```

Pour récupérer la valeur de la chaîne "label" basée uniquement sur son ID.

Depuis API 23 de Android, les méthodes de récupération de certains types de ressources comme les images (drawables) et les couleurs (colors) font désormais partie de la classe `ContextCompat`. Par exemple, la récupération d'une ressource image et couleur se fait de la manière suivante :

```
ContextCompat.getDrawable(getActivity(), R.drawable.MyImage);

ContextCompat.getColor(getActivity(), R.color.red);
```

CHAPITRE 4
L'interface utilisateur

Objectifs unitaires de l'enseignement :

- Savoir manipuler les différents éléments de l'interface graphique,
- Apprendre les écouteurs d'évènements,

1. Introduction

L'interface utilisateur de l'application est tout ce que l'utilisateur peut voir et interagir avec. Android fournit une variété de composants d'interface utilisateur qui permettent de créer l'interface utilisateur de l'application.

En effet, la définition de l'interface graphique pour une application mobile Android se fait majoritairement en XML. Cela est justifié par le fait que la définition XML de l'interface graphique soit plus souple que la définition Java qui requière de faire beaucoup d'instanciations et qui la partie code qui en résulte est volumineuse. Par conséquent, dans le code Java, on se suffit de la manipulation dynamique de l'interface graphique déjà définie en XML.

Dans ce chapitre, nous allons découvrir les aspects essentiels pour la conception et la gestion des différents styles d'interface utilisateur.

2. Les Views

Un View, appelé également Widget, représente l'objet graphique élémentaire que l'utilisateur peut voir et interagir avec. On distingue plusieurs types de Views à savoir le Bouton, le TextView, EditText, WebView, ImageView, etc. Dans ce qui suit, nous allons découvrir les exemples de définition XML des Views que nous trouvons dans la plupart des applications mobiles. Avant cela, il est important de mettre en évidence les propriétés communes entres les Views.

2.1. Propriétés communes

Les différents types de Views partagent des propriétés communes concernant leur apparition dans l'interface graphique de l'application. Cette partie est consacrée à l'explication de telles propriétés.

- **xmlns:android="http://schemas.android.com/apk/res/android":**
  Chaque partie XML spécifiant un View doit commencer par cette ligne. Il s'agit d'une déclaration d'espace de nom XML afin de préciser que les attributs du View (identifiant, dimensions,..) sont liés à Android.

- **Identifiant :** l'identifiant du View est spécifié à l'aide de l'attribut ''id''.

  En XML : **`android:id="@+id/identifiant_View"`**

- **Dimension :** on veut dire par dimension du View la largeur (width) et la hauteur (height) qui sont spécifiées à l'aide des deux attributs respectifs : ''layout_width'' et ''layout_height''.

  En XML : **`android:layout_width="fill_parent"/"match_parent"/`**
  `"wrap_content"/"valeur dp"`
  `android:layout_height="fill_parent"/"match_parent"/`
  `"wrap_content"/"valeur dp"`

  `fill_parent` indique que le View remplit toute la place dans le parent en largeur ou en hauteur. Sachant que le parent par rapport à un View c'est le Layout dans lequel il est défini. `Match_parent` indique plutôt que le view occupe l'espace restant dans le parent en largeur ou en hauteur. Finalement, le View ayant `wrap_content` comme valeur de largeur ou la hauteur occuperait seulement la place minimale nécessaire (en largeur ou hauteur).
  En effet, les attributs `layout_width` et `layout_height` peuvent prendre des valeurs numériques précisant la mesure exacte en dpi. Par exemple un View de type Bouton peut être configuré pour avoir les dimensions suivantes :
  `android:layout_width="200dp"` et
  `android:layout_height="40dp"`

- **Orientation :** définit l'orientation suivant laquelle les Views sont empilés dans le Layout. Elle peut être verticale ou horizontale.

  En XML : **`android:orientation="vertical"/"horizontal"`**

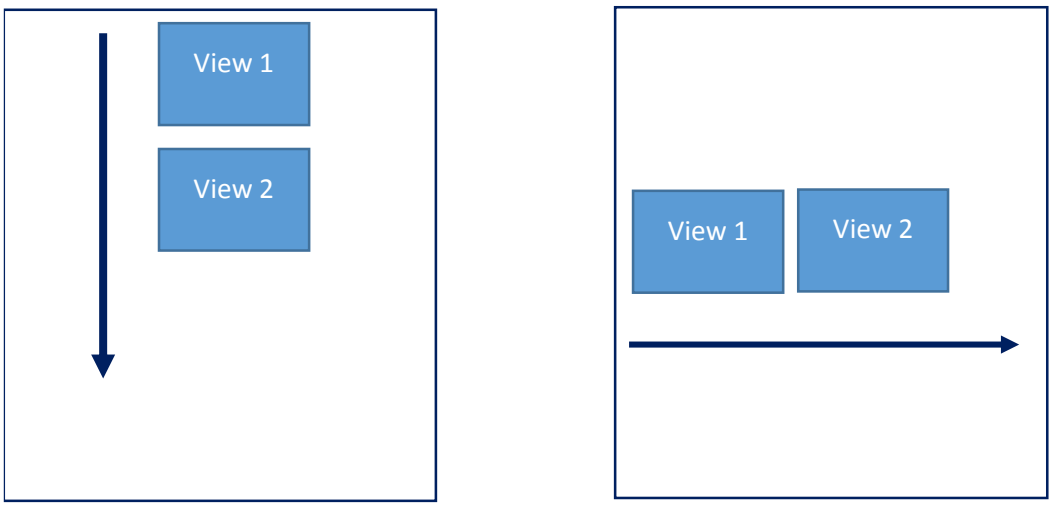


Figure 4.1. L'orientation verticale et horizontale des Views

- **Alignement :** pour l'alignement du View à gauche, à droite, au centre de son conteneur. Le View peut être également aligné au centre verticalement ou au centre horizontalement. Dans tous les cas, l'attribut ''gravity'' est utilisé.

En XML: `android:gravity="left"/"right"/"center"/`
`"center_vertical"/"center_horizontal"`

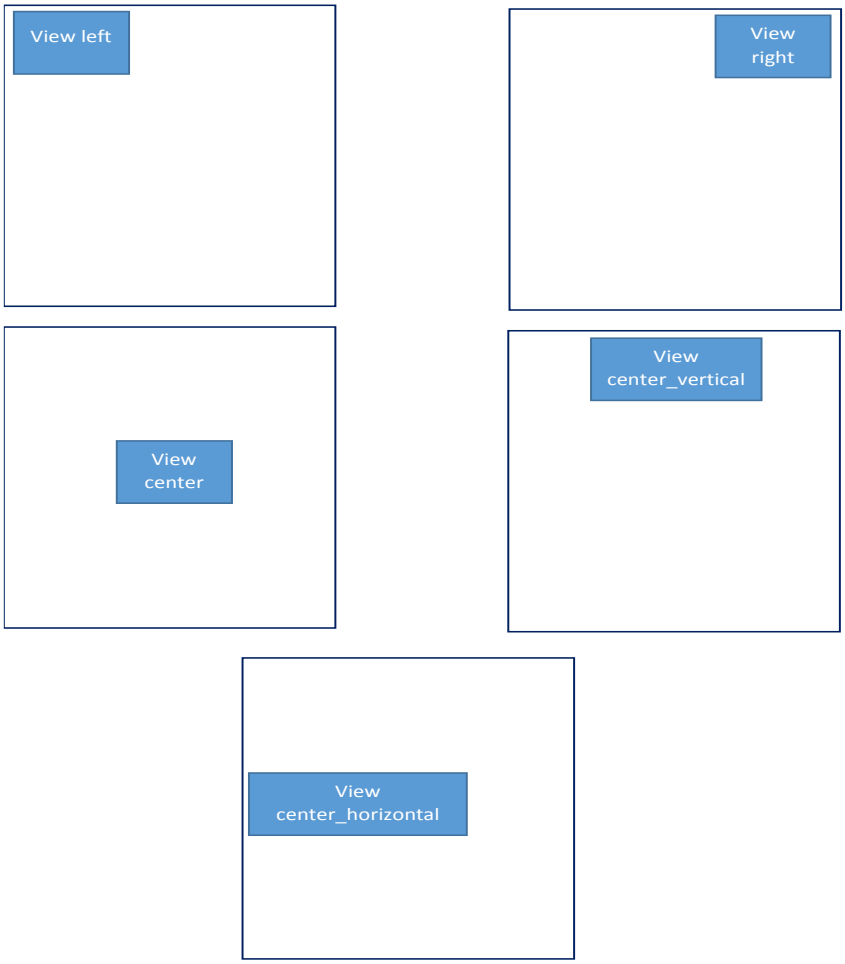


Figure 4.2. Les différents types d'alignement des Views.

- **Espacement :** utile pour gérer manuellement le positionnement du View dans l'interface (le Layout).

En XML : `android:paddingRight="valeur dp"`
`android:paddinLeft="valeur dp"`

Ou bien :

`android:marginRight="valeur dp"`
`android:marginLeft="valeur dp"`
`android:marginTop="valeur dp"`

```
android:marginBottom="valeur dp"
android:margin="memeValeur_TopBottomRightLeft dp"
```

- **Visibilité :** pour indiquer si le View doit être visible, invisible ou disparu.

  En XML : `android:visibility="visible"/"invisible"/"gone"`

  On peut par exemple configurer en XML un bouton pour qu'il soit visible au chargement de l'application puis, on le fait disparaître suite à un évènement. Dans ce cas, le bouton devient invisible dynamiquement dans le code Java à l'aide de la méthode `setVisibility(View.GONE);`

- **Texte explicatif :** utile pour les Views affichant un texte à l'utilisateur ou bien les Views qui sont destinés à contenir un texte saisi par l'utilisateur. Le texte explicatif est important pour tels types de Views afin de donner une idée à l'utilisateur sur ce que le View devrait contenir comme texte/données. Le texte explicatif prend la couleur grise.

  En XML : `android:hint="un text"`

2.2. Exemples de Views

A. Le bouton (Button)

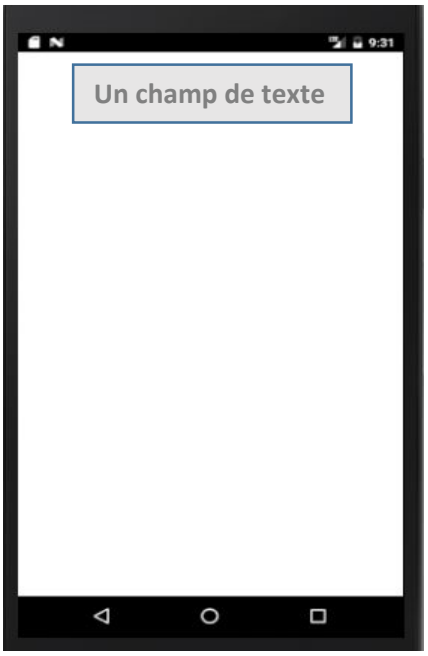
```
<Button
   xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button"
    android:text="OK"
    android:layout_width="wrap_parent"
    android:layout_height="wrap_parent"
    android:layout_gravity="center_horizontal"

    />
```

Graphiquement, on obtiendrait le résultat suivant :

B. L’étiquette de texte (TextView)

```
<TextView
   xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_parent"
    android:layout_height="wrap_content"
   endroid:gravity="center_vertical"
    android:hint="Un champ de texte"/>
```

Graphiquement, ça donnerait le résultat suivant :



C. Le champ de saisie (EditText)

```
<EditText
   xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/EditTXT"
    android:layout_width="wrap_parent"
    android:layout_height="wrap_parent"
    android:hint="Entrez votre texte!"
   android:gravity="center"
    android:singleLine="false"
    />
```

L'attribut `singleLine` est un attribut spécifique au View EditText et dans notre exemple, il porte la valeur "`false`". Cela veut dire qu'il serait possible de saisir plusieurs lignes de texte dans le champ de saisie. Si par contre on veut que le champ supporte seulement une seule ligne, il suffirait d'affecter la valeur "`true`" à l'attribut `singleline`.

Le résultat graphique de l'exemple serait donc :

D. Conteneur d'image (ImageView)

```
<ImageView
   xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/logo_univ"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:adjustViewBounds="true"
        android:src="@drawable/MyImages"
        />
```

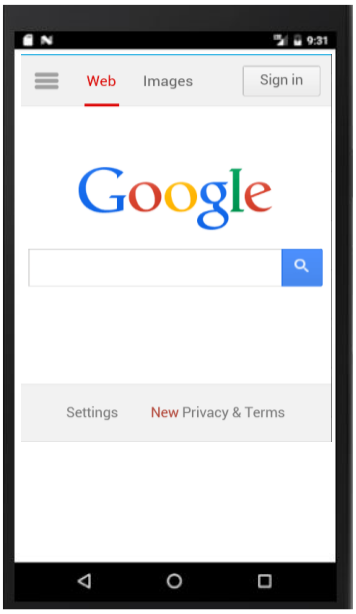
Dans cet exemple on veut charger une ressource image (le logo de l'université de Biskra) dans un ImageView. D'après l'exemple, l'imageView doit prendre tout l'espace en largeur et hauteur.

E. Le conteneur de pages web (WebView)

```
<WebView
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="700dp"
/>
```

Ex XML, On peut seulement créer le WebView. L'association de la page web à charger dans notre WebView se fait dans le code Java via la méthode :
`WebView.loadURL(String url);`
Par exemple, si on veut charger la page web https://www.google.com, on doit avoir dans le code : `objWebView.loadURL`("https://www.google.com ");

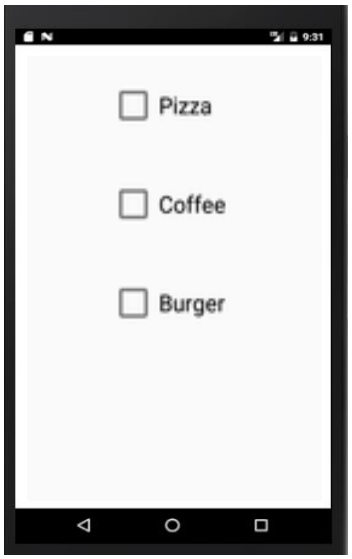
Le résultat graphique est :



F. Cases à cocher (CheckBox)

```
<CheckBox
   xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/check1"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="@string/Pizza" />
<CheckBox
   xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/check2"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="@string/Coffee" />
<CheckBox
   xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/check3"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="@string/Burger" />
```

Le résultat graphique serait :

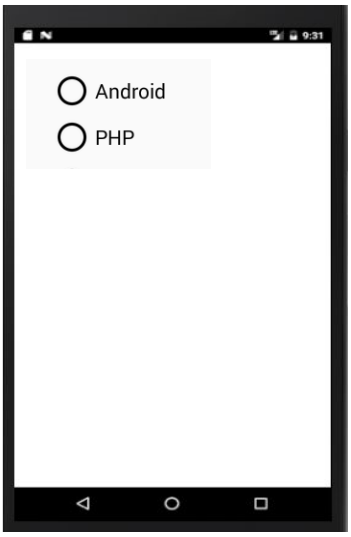


G. Le groupe de boutons radio (RadioGroup)

Le RadioGroupCe est un type de View qui a la forme d'un groupe de cases à cocher nommés RadioButton où la sélection est exclusive.

```
<RadioGroup
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_parent"
    android:layout_height="wrap_parent">
        <RadioButton android:id="@+id/radio1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Android" />
        <RadioButton android:id="@+id/radio2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="PHP" />
</RadioGroup>
```

Le résultat graphique de l'exemple est comme montré dans la figure suivante :



3. Le Layout

Le Layout définit la structure d'une interface utilisateur dans l'application et il dessine la vue graphique tel qu'elle doit s'afficher à l'utilisateur. Tous les éléments de la présentation sont créés à l'aide d'une hiérarchie d'objets View et ViewGroup. Le ViewGroup est un conteneur invisible qui comporte plusieurs Views et/ou sous ViewGroups, comme le montre la figure ci-dessous.

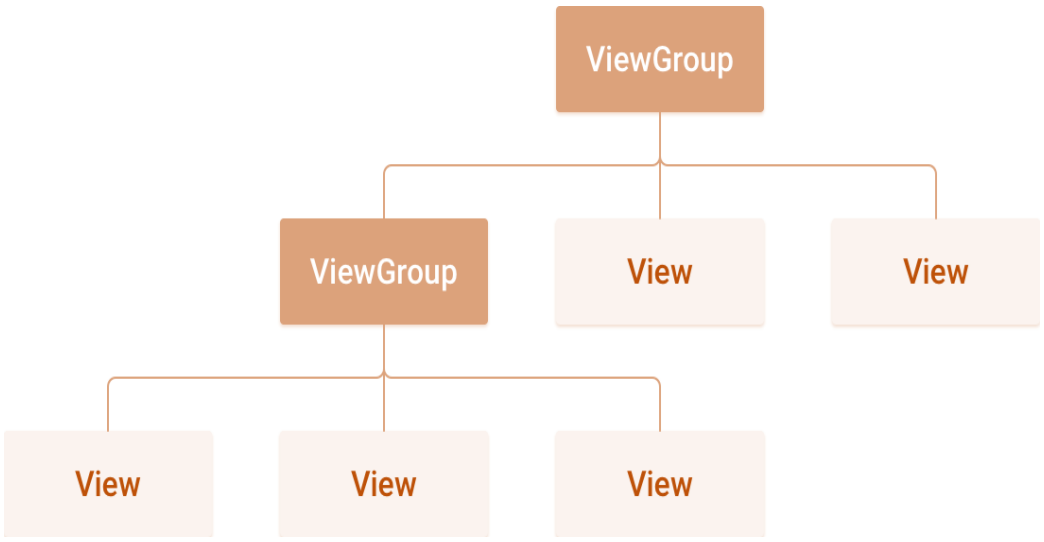


Figure 4.3. Structure générale d'un Layout (le ViewGroup).

A ce niveau, il est important de retenir que :

- Layout et ViewGroup représentent tous les deux l'interface graphique. Donc, qui dit layout, ou ViewGroup, dit également interface graphique.

- Le Layout définit le positionnement des Views.
- On avait précisé dans le chapitre précédent que l'activité c'est un conteneur de l'interface graphique. Suite à cela, on doit associer à chaque activité de l'application, le Layout qui est destiné à être affiché.

3.1. Les types de Layouts

On distingue plusieurs types de Layout. La différence clé entre les différents types réside dans le style suivant lequel le Layout va disposer les Views dans l'interface.

- **Le Layout linéaire (LinearLayout)**

Comme l'indique le nom, le Layout linéaire dispose les Views de façon linéaire verticalement ou horizontalement.

- **Le Layout relatif (RelativeLayout)**

Dans ce type de Layout, les Views sont placés les uns par rapport aux autres. Par exemple, dans un Layout de login, le bouton de validation des informations saisies doit être placé après deux champs de saisie (EditText) pour saisir l'adresse email et le mot de passe.

A partir de l'API 9, l'utilisation d'une variante sophistiquée du Layout relatif appelée ConstraintLayout est devenue possible. La variante permet de positionner et dimensionner les Views de manière plus flexible où plusieurs contraintes doivent être considérées à titre d'exemple, la gestion optimisée des marges, positionnement circulaire des Views, mise en chaine des Views avec différents dimensions, etc.

- **Le Layout Grid (GridLayout)**

Offre une disposition matricielle des Views tout comme Android présente les icones de toutes les applications.

3.2. Définition XML du Layout

Il faut créer un fichier XML dans le répertoire res/layout/. La définition XML du Layout est assez simple. Il suffit d'insérer tous les Views avec leurs configurations au sein du tag Layout sachant que les Layouts partagent eux aussi les propriétés communes avec les Views.

Le modèle général de définition du Layout en XML est :

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

       <EditText
            .....
            />
       <Button
            .....
            />
       <Textview
            .....
            />
        <WebView
            .....
            />
            .....
    </typeLayout>
     ============================================================
```

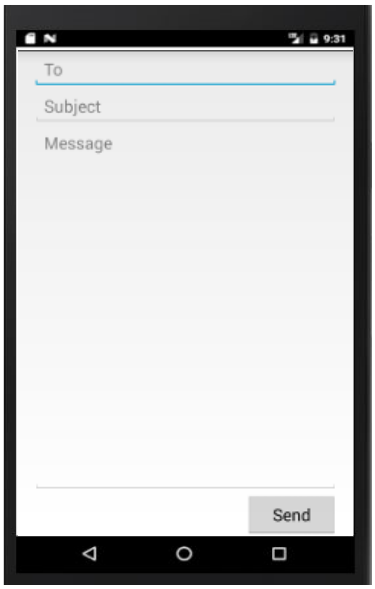
Un exemple de chaque type de Layout est donné dans ce qui suit :

- **Exemple de définition XML du Layout linéaire**

L'exemple correspond à une interface graphique pour envoyer un email. Le Layout est linéaire et il dispose les Views verticalement. Le Layout de l'exemple comporte quatre Views dont trois sont de type EditText et un Bouton.

```
<?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:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

L'interface graphique correspondante au Layout de l'exemple est comme montré dans la figure suivante :



- **Exemple de définition XML du Layout relative**

Les Views sont positionnés dans le Layout les uns par rapport aux autres.

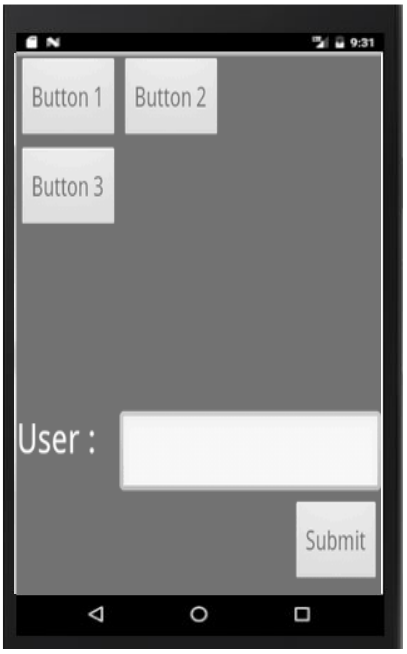
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <Button
        android:id="@+id/btnButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1"/>

    <Button
        android:id="@+id/btnButton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2"
        android:layout_toRightOf="@+id/btnButton1"/>

     <Button
```

```
        android:id="@+id/btnButton3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3"
        android:layout_below="@+id/btnButton1"/>

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/btnButton3"
        android:layout_marginTop="94dp"
        android:text="User :"android:textAppearance=
                        "?android:attr/textAppearanceLarge"
       />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignTop="@+id/textView1"
        android:layout_toRightOf="@+id/btnButton3" />

    <Button
        android:id="@+id/btnSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/editText1"
        android:text="Submit" />

</RelativeLayout>
```

❖ **Remarque**

Les attributs de type: `android:layout_below` et `android:layout_toRightOf` sont particulièrement importants dans les Layout relatifs car ils permettent de placer des Views par rapports à d'autres Views déjà définis dans le même Layout relatif.

Graphiquement, le Layout relatif de l'exemple serait :



- **Exemple de définition XML du Layout matriciel**

L'exemple présente une calculatrice simple où on a besoin de boutons positionnés de façon matricielle. Pour ce faire il faut une gestion des positions sur les lignes et les colonnes avec les attributs :

`android:layout_column, android:layout_columnSpan, android:layout_columnWeight,android:layout_row, android:layout_rowWeight`

La définition XML de l'exemple du Layout matriciel correspondant à une interface de calculatrice simple est :

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="4"
```

```
android:rowCount="4">

<Button
    android:layout_column="0"
    android:layout_columnWeight="1"
    android:layout_row="0"
    android:layout_rowWeight="1"
    android:text="7" />

<Button
    android:layout_column="1"
    android:layout_columnWeight="1"
    android:layout_row="0"
    android:layout_rowWeight="1"
    android:text="8" />

<Button
    android:layout_column="2"
    android:layout_columnWeight="1"
    android:layout_row="0"
    android:layout_rowWeight="1"
    android:text="9" />

<Button
    android:layout_column="3"
    android:layout_columnWeight="1"
    android:layout_row="0"
    android:layout_rowWeight="1"
    android:text="x" />

<Button
    android:layout_column="0"
    android:layout_columnWeight="1"
    android:layout_row="1"
    android:layout_rowWeight="1"
    android:text="4" />
```

```
<Button
    android:layout_column="1"
    android:layout_columnWeight="1"
    android:layout_row="1"
    android:layout_rowWeight="1"
    android:text="5" />

<Button
    android:layout_column="2"
    android:layout_columnWeight="1"
    android:layout_row="1"
    android:layout_rowWeight="1"
    android:text="6" />

<Button
    android:layout_column="3"
    android:layout_columnWeight="1"
    android:layout_row="1"
    android:layout_rowWeight="1"
    android:text="-" />

<Button
    android:layout_column="0"
    android:layout_columnWeight="1"
    android:layout_row="2"
    android:layout_rowWeight="1"
    android:text="1" />

<Button
    android:layout_column="1"
    android:layout_columnWeight="1"
    android:layout_row="2"
    android:layout_rowWeight="1"
    android:text="2" />

<Button
    android:layout_column="2"
```

```
        android:layout_columnWeight="1"
        android:layout_row="2"
        android:layout_rowWeight="1"
        android:text="3" />

    <Button
        android:layout_column="3"
        android:layout_columnWeight="1"
        android:layout_row="2"
        android:layout_rowWeight="1"
        android:text="+" />

    <Space
        android:layout_column="0"
        android:layout_columnWeight="1"
        android:layout_row="3"
        android:layout_rowWeight="1" />

    <Button
        android:layout_column="1"
        android:layout_columnWeight="1"
        android:layout_row="3"
        android:layout_rowWeight="1"
        android:text="0" />

    <Button
        android:layout_column="2"
        android:layout_columnSpan="2"
        android:layout_columnWeight="1"
        android:layout_row="3"
        android:layout_rowWeight="1"
        android:text="=" />

</GridLayout>
```

Le résultat graphique de l'exemple est montré dans la figure ci-dessous :

3.3. Association du layout à l'activité

Après avoir défini l'interface graphique dans un fichier ''nomLayout.xml'', dans le répertoire res/layout/ l'interface doit être associée à l'activité pour qu'elle puisse être affichée à l'utilisateur. L'association se fait dans le code Java à l'aide de la méthode `setContentView(R.layout.nomLayout);` au sein de la méthode callback onCreate() de l'activité comme suit :

```
============================================================
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.nomlayout);
}
============================================================
```

❖ **Remarque :**

Au cas où le développeur manque cette étape, rien ne va s'afficher dans l'activité qui est destinée à contenir l'interface.

4. Manipulation de l'interface dans le code java

Dans le code Java, il est possible de manipuler les éléments de l'interface graphique de l'application et de modifier leur configuration. Cela est fait à l'aide d'un ensemble de méthodes spécifiques. Dans cette section nous allons mettre la lumière sur les méthodes les plus essentielles et les plus usuelles.

- **Récupération d'une instance d'un View déjà défini en XML dans le code Java à partir de son identificateur :**

```
View objet = findViewById(R.id.nom_View);
```

Voici quelques exemples :

```
Button b = (Button) findViewById(R.id.MyButton);
TextView tv = (TextView) findViewById(R.id.TxtV1);
EditText editObj = (EditText) findViewById(R.id.MyButton);
```

- **Ajout ou récupération de texte à partir du View**

```
objet_View.setText(String txt);
String s = objetView. getText();
```

Voici un exemple d'utilisation avec un textView :

```
TextView tv = (TextView) findViewById(R.id.TxtV1);
tv.setText("hello") ;
```

L'exemple ci-dessus illustre l'affectation de la chaine ''hello'' à un TextView.

```
EditText editObj = (EditText) findViewById(R.id.MyButton);
String s = editObj.getText();
Tv.setText(s;
```

Dans cet exemple, on a plutôt affecté la chaine contenue dans l'EditText dans le TextView.

❖ **Remarque :**

Il y a d'autres méthodes qui peuvent être utilisées pour changer en Java la position du View, sa couleur, la taille du texte. Cependant puisque il s'agit des spécificités statiques, il est recommandé de les définir et les modifier en XML.

5. Quelques composants graphiques avancés

Dans cette partie, nous abordons deux types d'objets graphiques avancés : les Menus et les ListViews.

5.1. Les Menus

Les menus sont un composant d'interface utilisateur courant dans de nombreux types d'applications. Les menus sont particulièrement utiles pour présenter des actions que l'utilisateur puisse entreprendre, ainsi que d'autres options dans les activités de l'application.

À partir d'Android 3.0 (API 11), les appareils Android ne sont plus tenus de fournir un bouton de menu dédié. Avec ce changement, les applications Android devraient s'éloigner d'une dépendance au panneau de menu traditionnel à 6 éléments et fournir à la place une barre d'applications pour présenter les actions courantes des utilisateurs.

- **La définition XML du menu**

```
============================================================
<typeLayout .....>

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/id_Item"
          android:title="@string/title_Item" >
    <item android:id="…"
          android:title="…" />
     <item android:id="…"
               android:title="…" />
</menu>

</typeLayout>
============================================================
```

L'élément **<item>** supporte plusieurs attributs que nous pouvons utiliser pour définir l'apparence et le comportement d'un élément. Les éléments du menu incluent les attributs suivants :

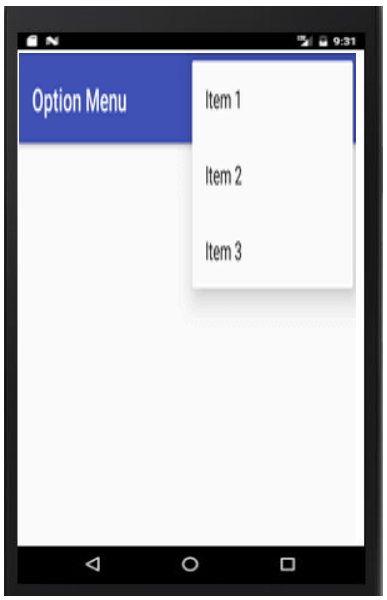
- **android:id :** Un ID de ressource unique à l'élément, qui permet à l'application de reconnaître l'élément lorsque l'utilisateur le sélectionne.
- **android : icône :** Une référence à un dessin à utiliser comme icône de l'élément.
- **android: titre :** Une référence à une chaîne à utiliser comme titre de l'élément.
- **android: showAsAction :** Spécifie quand et comment cet élément doit apparaître en tant qu'élément d'action dans la barre d'applications. La valeur e cet attribut peut être :
  - **withText :** Inclut également le texte du titre (défini par android: title) avec l'élément d'action.
  - **ifRoom :** L'élément n'est placé dans la barre d'applications que s'il y a de la place pour lui.

- **never :** l'élément ne doit pas apparaître dans la barre d'applications.
- **always :** Placez toujours cet élément dans la barre d'applications. Évitez de l'utiliser sauf s'il est essentiel que l'élément apparaisse toujours dans la barre d'action. Si vous définissez plusieurs éléments pour qu'ils apparaissent toujours comme des éléments d'action, ils peuvent se chevaucher avec une autre interface utilisateur dans la barre d'applications.

❖ **Exemple :**

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item  android:id="@+id/item1"
        android:title="Item 1"/>
    <item  android:id="@+id/item2"
        android:title="Item 2"/>
    <item  android:id="@+id/item3"
        android:title="Item 3"
        android:showAsAction="withText"/>
</menu>
```

- Résultat graphique :



❖ **Remarque :**

Dans on veut définir un sous menu, il faudrait intégrer un sous tag **<menu>** à l'intérieur du tag **<item>** correspondant à l'élément auquel le sous menu serait associé.

```
==========================================================
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/id_Item"
          android:title="@string/title_Item" >
        <menu>
            <item android:id="…"
                  android:title="…" />
            <item android:id="…"
                  android:title="…" />
         </menu>
    </item>
</menu>
 ==========================================================
```

5.2. Les ListViews

Un ListView est un type spécial de view regroupant un ensemble de views de même type, tout comme les boutons radio, avec l'exception que les éléments de la liste soient gérés dynamiquement par un adaptateur (ArrayAdapter)

- **Etapes pour définir une liste en Java**

- Dans le code de l'activité, il faut d'abord définir un tableau de String contenant les éléments que nous voulons voir sur notre liste.
- Créer une instance de la classe `ArrayAdapter<String>` en lui attribuant le tableau de String déjà créé.
- Associer la liste (objet ListView) avec l'objet `ArrayAdapter<String>` à l'aide de la méthode `setAdapter();`

❖ **Exemple :**

```
public class MainActivity extends Activity {
    ListView MylistView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        MylistView = (ListView) findViewById(R.id.listview);

        String[] listeCourse = new String[]{
                "Red",
                "Yellow",
                "Blue",
                "Orange",
                "Black",
                "Green",
                "Brown",
                "Grey"
           };

         ArrayAdapter<String> MyArrayAdapter = new
             ArrayAdapter<String>(this,
              android.R.layout.exemple,
             listeCourse);

        MylistView.setAdapter(MyArrayAdapter);

}
```

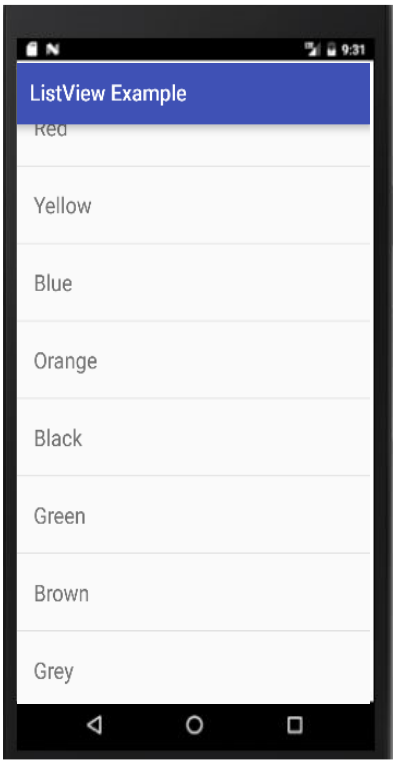
- **Définition XML du ListView**

==========

```
<typeLayout……>
<ListView
    android:id="@+id/listviewID"
    android:layout_width="match_parent|fill_parent|wrap_content"
    android:layout_height="match_parent|fillparent|wrap_content">
</ListView>
</typeLayout>
```

==========

- Le résultat graphique :



6. Les écouteurs d'évènements

Pour gérer les interactions de l'utilisateur avec les Views de l'interface graphique, Android met à la disposition des développeurs les écouteurs d'évènements (*event listeners*). On sous-entend par évènement : un clic sur le View, un clic long, une touche, un changement du texte, etc.

Un écouteur d'évènement comporte une méthode callback à surcharger pour spécifier le comportement et les actions à faire pour répondre à l'apparition d'un certain évènement. Le tableau ci-dessous illustre quelques méthodes gestionnaires d'évènement avec les classes.

Table 4.1. Exemples d'écouteurs d'évènements

Méthode callback (gestionnaire d'évènement)	Classe de l'écouteur
`public void onClick(View v)`	`View.OnClickListener`
`public void onLongClick(View v)`	`View.OnLongClickListener`
`public void onTouch(View v)`	`View.OnTouchListener`
`public void onMenuItemClick(View v)`	`View.OnMenuItemClickListener`
`public void onTextChanged(View v)`	`View.OnTextChangedListener`

- **Remarque :**

Les méthodes callback gérant les évènements sur les Views doivent obligatoirement avoir la signature indiquée dans le tableau.

L'enregistrement de l'écouteur d'évènement se fait généralement dans le code java de l'activité en respectant les étapes suivantes :

- Instanciation d'un objet écouteur
- Redéfinition de la méthode callback pour indiquer ce qui doit se faire pour gérer l'évènement.
- Enregistrement de l'écouteur en associant l'objet écouteur à l'objet View.

Ces étapes peuvent être introduites séparément ou incluses l'une dans l'autre, dans le code. De plus, pour certains types d'écouteur (onClick), il est possible de faire l'enregistrement de l'évènement en XML, exactement dans la partie définissant les propriétés du View concerné par l'évènement. L'enregistrement se réaliserait donc à travers l'attribut `anroid:oncCick= "nom_Methode"` et dans le code java on définit la méthode avec la signature suivante :

```
public void nom_Methode(View v).
```

Le corps de la méthode serait le comportement de gestion de l'apparition de l'évènement.

❖ Exemple d'utilisation

L'exemple montre comment gérer un clique sur bouton et y réagir en affichant un Toast. Supposant donc une application ont l'activité principale comporte un bouton. Quand l'utilisateur clique dessus, un Toast s'affiche.

- **Le contenu du fichier res/layout/main_layout.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical "
    >
    <Button
       android:id="@+id/button"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="cliquez ici !"
       android:gravity="center"
      />

</LinearLayout>
```

- **Le code de l'activité :**

```
public class MainActivity extends Activity {
   Button b;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main_layout);
      b = (Button)findViewById(R.id.button);

     //Enregistrement de l'écouteur de l'évènement
     b.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            //définir ici le comportement
            Toast.makeText(getApplicationContext(),"Bouton
                 clique!",Toast.LENGTH_SHORT).show();
         }
      });
   }  }
```

- Le résultat graphique avant et après le clique sur bouton :

- **Enregistrement XML de l'écouteur onClickListener :**

Dans la partie `<Button/>`, il faut ajouter la ligne :

`android:onClick="Handler_meth"`

Et dans l'activité on définit la méthode `public void Handler_meth(View v) ;`

```
<Button
          android:id="@+id/button"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="cliquez ici !"
          android:gravity="center"
          android:onClick="Handler_meth"

         />
```

```
public class MainActivity extends Activity {
   Button b;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main_layout);
```

```
      b = (Button)findViewById(R.id.button);
    }
    public void Handler_meth (View v) {
        Toast.makeText(getApplicationContext(),"Bouton
        cliqué!",Toast.LENGTH_SHORT).show();
    }
}
```

CHAPITRE 5

Les Intents et les Broadcast receivers

Objectifs unitaires de l'enseignement :

- Apprendre à manipuler les Intents.
- Maitriser les Intents explicites et les Intents implicites
- Apprendre à programmer les Broascast Receivers.

1. Les intents

Les intents forment un composant clé dans les applications mobiles Android. Avec un objet Intent nous pouvons faire communiquer les différents composants de l'application (par exemple une activité et un récepteur de diffusion (broadcast receiver), une activité et un service, une activité et une autre activité de la même application). En effet, différents types 'actions pouvant être réalisées à partir de l'application comme : envoi d'un email l'appel téléphonique, envoi d'un SMS, etc. sont lancées par des Intents.

1.1. Les composants d'un intent

Un intent contient principalement les informations suivantes :

- **L'action à réaliser :** sous forme de chaine de caractère
  Les actions peuvent être natives (prédéfinies par le système) ou définies par le développeur. Voici quelques exemples d'actions natives :

  – android.intent.action.CALL appel téléphonique
  – android.intent.action.EDIT affichage de données pour édition par l'utilisateur
  – android.intent.action.MAIN activité principale d'une application
  – android.intent.action.VIEW affichage de données
  – android.intent.action.WEB_SEARCH recherche sur le WEB

- **Les données** sur lesquelles l'action va porter (contenu MIME ou URI)

- **La catégorie** pour spécifier la catégorie de l'action demandée.
  Voici quelques exemples e catégories :

  – android.intent.category.DEFAULT activité pouvant être lancée explicitement
  – android.intent.category.BROWSABLE peut afficher une information désignée par un lien
  – android.intent.category.LAUNCHER activité proposée au lancement par Android

- **Les extras :** donnés supplémentaires nécessaires pour accomplir l'action. Par exemple pour l'envoi d'un email, l'action c'est ACTION_SEND, les données c'est l'adresse email, les extras sont l'objet du mail et le message à envoyer.

1.2. Les types des intents

Les intents qui sont utilisés pour déclencher les activités, peuvent être implicites ou explicites.

- **Intent Explicite :** Spécifie les composants qui précisent la classe exacte qui doit être exécutée.

- Syntaxe de création :

```
Intent intentExplicite = new Intent(context, nomClass.class);
```

- **Intent Implicite :** Ne spécifie pas le composant mais fournit assez d'informations permettant au système de déterminer les composants nécessaires correspondant à cette action.

- Syntaxe de création :

```
Intent intentImplicite = new Intent(String action, Uri data);
```

- Ou bien création avec action seule :

```
Intent intentImplicite = new Intent(String action);
```

1.3. Démarrage d'une activité à l'aide d'un intent

Suivant que le type de l'intent soit implicite ou explicite, le démarrage d'une activité à l'aide d'un intent est ainsi dit implicite ou explicite. La méthode à évoquer est :

```
startActivity(intent);
```

- **Exemple de démarrage implicite :** Appeler un numéro de téléphone :

```
Uri numero = Uri.parse("tel:0123456789");
// Il faut utiliser la méthode parse() ; de la classe Uri pour mettre les données au
//format URI.
Intent appeler = new Intent (Intent.ACTION_CALL, numero);
startActivity(appeler);
```

- **Exemple de démarrage explicite :** Démarrer l'activité qui s'occupe d'un appel téléphonique.

```
Intent appeler = new Intent(getApplicationContext(),
                                        GivePhoneNumber.class);
```

```
startActivity(appeler);
```

1.4. Les méthodes de la classe Intent

En plus es méthodes e création de l'objet intent implicite ou explicite), la classe intent fournit d'autres méthodes utiles :

- **`getIntent();`** retourne l'objet intent qui a démarré l'activité.
- **`getData();`** permet de récupérer les données de l'intent
- **`getAction();`** retourne l'action associée à l'intent.
- **`addCategory(String category);`** ajoute une catégorie à l'intent
- **`setDataAndType(Uri data, String type);`** définit l'Uri et le type mime pour les données.

1.5. Ajout de données à un intent

Deux façons pour ajouter des données supplémentaires (Extra data) à l'intent

A. Ajout direct :

Dans ce cas, il faut utiliser la méthode `putExtra()` autant de fois que nécessaire.

```
objetIntent.putExtra(''clé'', valeur);
```

La clé permet de repérer la donnée, chose qui faciliterait son récupération.

B. Ajout via objet Bundle :

Dans le cas où on a plusieurs données à ajouter à l'Intent, il serait plus flexible de passer par un objet *Bundle*. On ajoute toutes les données à l'objet Bundle puis, on associe ce dernier à l'objet Intent, une bonne fois pour toutes.

```
Bundle extras = new bundle(); //créer l'objet Bundle
//ajout des données à l'objet Bundle
extras.putString(''clé1'',''chaine de caractères'');
extras.putInt(''clé2'', valeur entière);
extras.putIntArray(''clé3'', [val1,val2,val3,...]);
//associer l'objet Bundle (extras) à l'objet Intent
objetIntent.putExtras(extras);
```

La figure ci-dessous récapitule et schématise les deux méthodes d'ajout de données à un objet Intent.

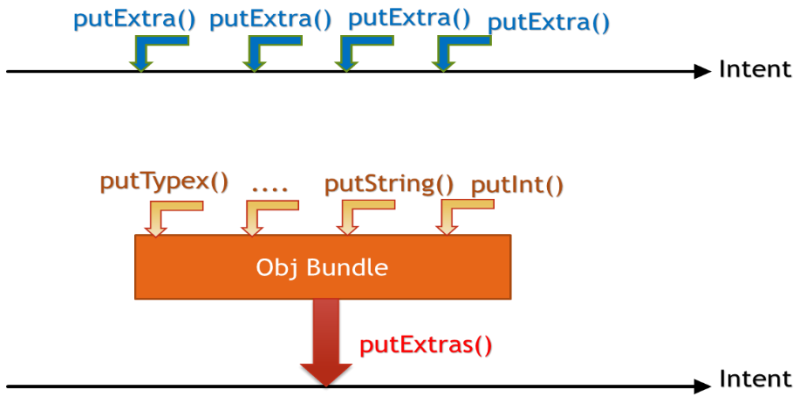


Figure 5.1. Deux méthodes d'ajout es données extra (avec et sans objet Bundle).

1.6. Récupération des données associées à un intent

Deux façons pour récupérer les données associées à l'Intent ayant démarré l'activité :

A. Le cas où les données ont été ajoutées directement

```
Intent i = getIntent();
//pour récupérer une donnée de type chaine de caractères
String s = i.getStringExtra(''clé'');
//pour récupérer une donnée de type entier
int x = i.getIntExtra(''clé'');
//pour récupérer une donnée de type réel
float f = i.getFloatExtra(''clé'');
......
```

B. Le cas où les données ont été ajoutées via objet Bundle

```
Intent i = getIntent();
Bundle extras = i.getExtras();
String s = extras.getString(''clé'');
Integer e = extras.getInt(''clé'');
.......
```

1.7. Réception et filtrage des intents

Android dispose d'un système de filtrage qui devrait être défini dans le fichier AndroidManifest.xml. Un filtre permet à une application de préciser et sélectionner les intents qu'elle veuille recevoir. Ainsi, un filtre peut utiliser plusieurs niveaux de filtrage :

- **Filtrage sur action :** identifie l'intent. Pour éviter les collisions, il est recommandé de respecter la convention de nommage java.
- **Filtrage sur catégorie :** filtrer une catégorie d'actions (DEFAULT, BROWSABLE, etc.)
- **Filtrage sur données :** texte ou multimédia.

Le filtrage se fait à l'aide de la balise <itent-filter> dans **AndroidManifest.xml** comme indiqué ci-dessous :

```
==========================================================
<manifest …….>
…….
<activity android:name=".ExampleActivity" >
  <intent-filter>
      <action android:name="android.intent.action.MAIN" />
     <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
   <intent-filter>
        <action android:name="android.intent.action.SEND" />
      <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
……..
</manifest>
==========================================================
```

- **Exemple :** Supposons une application ayant deux activités. L'activité principale comporte un bouton quand l'utilisateur clique dessus, la deuxième activité est lancée tout en lui passant une valeur entière quelconque à afficher dans un TextView.

- **Le code java de l'activité principale**

```
public class MainActivity extends Activity {
    Button btn;
    Intent I = new Intent(this, SecondAct.class);
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn = (Button)findViewById(R.id.btnClick);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                   I.putExtra("val",7);
                   startActivity(I);
            }
        });
  }
}
```

- **Le code java de la deuxième activité**

```
public class SecondActivity extends Activity {
    TextView tView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tView = (TextView)findViewById(R.id.txtResult);
        Intent I = getIntent();
        int x = I.getIntExtra("val");
        tView.setText("Start value :"+x);
    }
}
```

- **Le layout de l'activité principale**

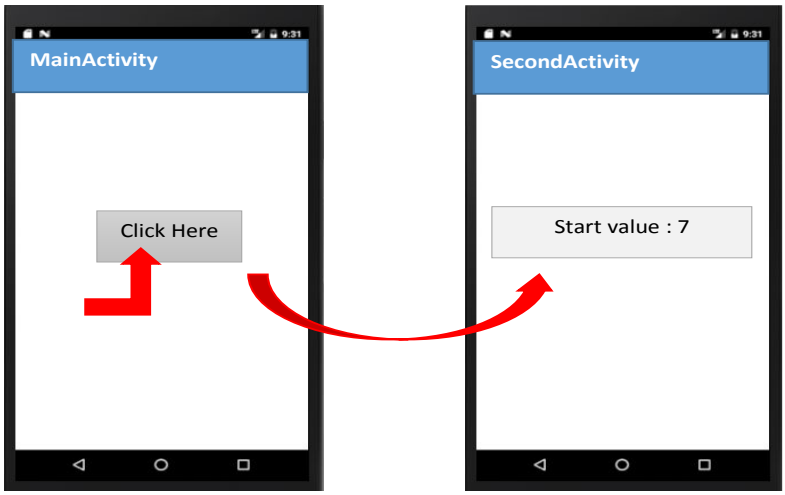
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:gravity="center" >
    <Button
```

```
        android:id="@+id/btnClick"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Here" />
</LinearLayout>
```

- **Le layout de la deuxième activité**

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

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

- **Le résultat graphique**



2. Les broadcast receivers

Le récepteur de diffusion (ou Broadcast receiver) est un composant Android qui permet à l'application mobile de s'inscrire aux événements système ou évènements applicatifs. Toutes les applications enregistrées à un événement sont averties par le système Android dès que cet événement se produise.

Lorsque l'un des événements en question se déclenche, le broadcast receiver met l'application, qui se déclare intéressée par l'évènement, en action. L'application dans ce cas, génère une notification ou effectue une certaine tâche à définir par le développeur.

Par exemple, les applications peuvent s'inscrire à l'événement système **ACTION_BOOT_COMPLETED** qui est déclenché une fois que le système Android a terminé le processus de démarrage (le boot).

Voici quelques autres exemples des actions les plus fréquemment générées par le système Android.

- **android.intent.action.BATTERY_LOW :** indiquer la condition faible batterie sur le telephone.
- **android.intent.action.DATE_CHANGED :** la date a été changée.
- **android.intent.action.REBOOT :** reboot
- **android.net.conn.CONNECTIVITY_CHANGE :** le réseau du mobile ou l'état de la connection wifi a été changé.

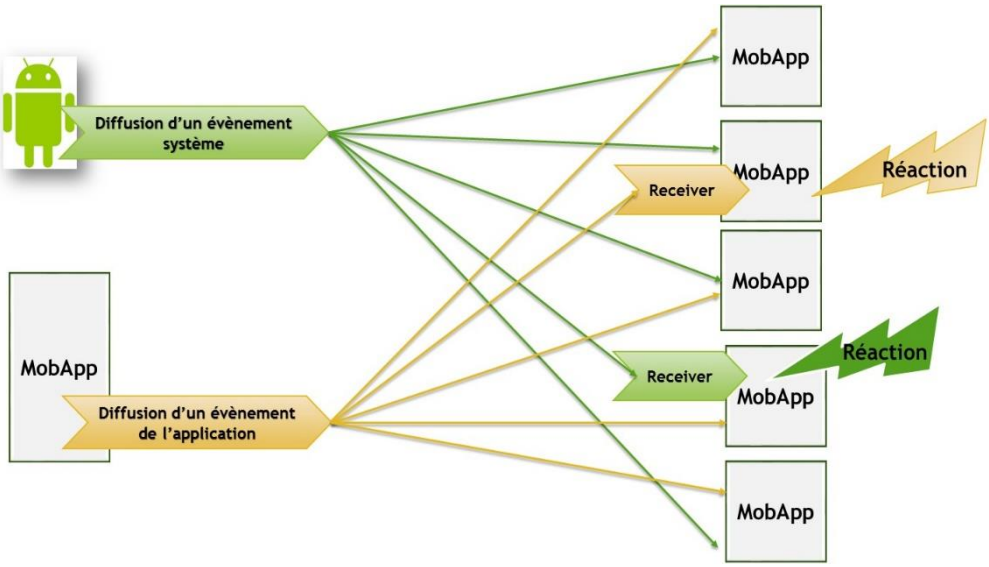


Figure 5.2. Principe de fonctionnement des broadcast receivers.

- **Remarque :**
  Contrairement aux activités, le Broadcast Receiver ne fournit aucune interface utilisateur.

2.1. Les étapes de création d'un broadcast receiver

On distingue deux étapes obligatoires pour définir un Broadcast Receiver : la création et l'enregistrement. Optionnellement, le développpeur peut expliciter le désenregistrement du récepeteur.

2.1.1. La création

Pour la création, il faut définir une classe héritant de la classe BroadcastReceiver et qui surcharge la méthode callback onReceive() comme montré dans l'exemple ci-dessous :

```
============================================================
public class MyReceiver extends BroadcastReceiver {
   public void  MyReceiver() {  //constructeur    }
       @Override
   public void onReceive(Context context, Intent intent) {
          //Ici l'application définit son comportement à la réception de l'intent de
          //l'évènement diffusé.
         //Dans cet exemple, on veut afficher l'action à l'apparition de l'évènement .
        Toast.makeText(context, "Action: " + intent.getAction(),
                                Toast.LENGTH_SHORT).show();
      }    }
============================================================
```

2.1.2. L'enregistrement

Pour l'enregistrement, un broadcast receiver peut être enregistré de deux manières.

A. Définition statique : En le définissant dans le fichier AndroidManifest.xml comme indiqué ci-dessous. Dans tel cas, un filter doit être inséré dans le tag receiver. Notons bien que les permissions relatives à l'évènement à capturer doivent etre déclarées dans le manifest. Faute de quoi la réception va échouer.

```
==========================================================
   <manifest.....>
   <application >
        <uses-permission
        android:name="android.permission.INTERNET" />
         <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE" />
        <receiver android:name = ".MyReceiver">
            <intent-filter>
                    <action android: name =
                  "android.net.conn.CONNECTIVITY_CHANGE" />
            </intent-filter>
        </receiver>
        ....
   </application>
   </manifest>
==========================================================
```

B. Définition dynamique : Alternativement à l'enregistrement statique, on peut enregistrer un Broadcast Receiver dynamiquement via la méthode : registerReceiver() dans le code de l'activité.

```
==========================================================
   IntentFilter filter = new IntentFilter();
   filter.addAction(getPackageName()+
   "android.net.conn.CONNECTIVITY_CHANGE");
   MyReceiver myReceiver = new MyReceiver();
   registerReceiver(myReceiver, filter);
==========================================================
```

2.1.3. Le désenregistrement

Pour désinscrire un Broadcast Receiver dans `onStop()` ou `onPause()` ou `onDestroy()` de l'activité, Il faut appeler la méthode `unregisterReceiver();`

```
============================================================
   protected void onPause() {
         unregisterReceiver(myReceiver);
         super.onPause();
    }
============================================================
```

2.2. La diffusion des évènements de l'application

Comme déjà révélé, le broadcast receiver est destiné à recevoir aussi bien des intents générés par le système Android que des Intents générés par les applications. Donc il est possible qu'une application diffuse un Intent pour toutes les autres applications. Cela devrait se faire à l'aide de la méthode `sendBroadcast()` depuis l'activité.

```
============================================================
   Intent intent = new Intent("com.example.anyUserAction_INTENT");
   sendBroadcast(intent);
============================================================
```

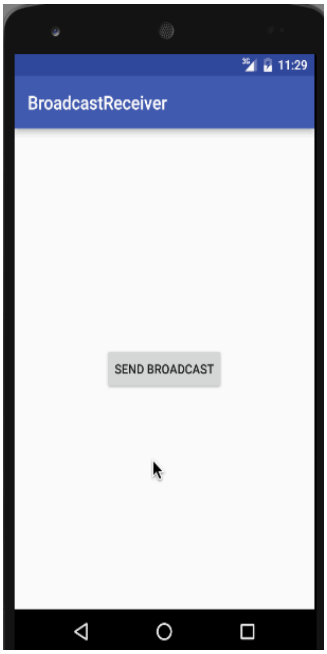
Il faut absolument définir cette action via un intent filter d'un tag receiver, dans le manifest ou faire le tout dans le code java de l'activité.

❖ Exemple d'utilisation

L'exemple consiste en une application qui nécessite -par supposition- la connexion Internet.

-L'application définit un Broadcast receiver pour intercepter les évènements de type changement de l'état de la connectivité réseau (internet ou autre).

-L'application a également la capacité de faire une diffusion d'un message en cliquant sur le bouton ''SEND BROADCAST''. L'activité principale de l'application est comme montré dans la figure ci-dessous.



-Dans tous les cas, un Toast doit afficher l'évènement survenu (clique sur bouton ou changement de l'état de la connexion wifi) à l'utilisateur.

- **Le fichier AnroiManifest.xml de l'application :**

```
<?xml version="1.0" encoding="utf-8"?>

<manifest
xmlns:android="https://schemas.android.com/apk/res/android"

package="com.example.broadcastreceiver">

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

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

<application

android:allowBackup="true"

android:label="@string/app_name">

<activity android:name=".MainActivity">

   <intent-filter>

      <action android:name="android.intent.action.MAIN" />

       <category
        android:name="android.intent.category.LAUNCHER" />
```

```
   </intent-filter>
</activity>
 <receiver android:name=".ConnectionReceiver">
    <intent-filter>
          <action
          android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
 </receiver>
</application>
</manifest>
```

- **Code complet de l'activité principale ''MainActivity.java''**

```
public class MainActivity extends Activity {
    ConnectionReceiver receiver;
    IntentFilter intentFLTR;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
         Button b = findViewById (R.id.BoutonDiffusion);
            receiver = new ConnectionReceiver();
      //l'association de l'action SOME_ACTION à l'intent filter, ainsi que le broadcast
      //receiver correspondant à l'action sont définis via le code java
              intentFLTR = new  IntentFilter
                   ("com.example.broadcastreceiver.SOME_ACTION");
    }
```

```
    @Override
    protected void onResume() {
         super.onResume();
         registerReceiver (receiver, intentFLTR);
    //l'action CONN_CHANGED est déclarée en xml (dans le manifest de
    //l'application)
    }
    @Override
    protected void onDestroy() {
          super.onDestroy();
    //l'application se désabonne de la réception des évènements à intercepter par le receiver
     unregisterReceiver(receiver);
    }
    public void diffusion(View v) {
    //l'application fait la diffusion du message de l'évènement (ou action) lors du
    //clique sur bouton
        Intent intent = new Intent
              ("com.example.broadcastreceiver.SOME_ACTION");
         sendBroadcast(intent);
    }
}
```

- **Le contenu du layout ''MainActivity.xml'' :**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="https://schemas.android.com/apk/res/android"
    xmlns:tools="https://schemas.android.com/tools"
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
    tools:context="com.example.broadcastreceiver.MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/BoutonDiffusion"
        android:text="Send Broadcast"
        android:gravity="center"
        />
</LinearLayout>
```

- **Code de la classe du Broadcast Receiver ‘’ConnectionReceiver.java’’**

```
public class ConnectionReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
     if (intent.getAction().
             equals("com.example.broadcastreceiver.SOME_ACTION"))
     { //Si l’évènement associé à l’Intent correspond à ’’SOME_ACTION’’,l’application
       //intéressée par l’apparition de l’évènement affiche un Toast
          Toast.makeText(context, "SOME_ACTION is received",
                  Toast.LENGTH_LONG).show();
      }
       else {
         //Sinon, il s’agit certainement d’un évènement système
        //‘’CONNECTIVITY_CHANGE’’,l’application intéressée par l’apparition
        //de l’évènement affiche un Toast indiquant l’état de changement de
        //connectivité réseau.
```

```
        ConnectivityManager cm = (ConnectivityManager)
    context.getSystemService(Context.CONNECTIVITY_SERVICE);

        NetworkInfo activeNetwork = cm.getActiveNetworkInfo();

        boolean isConnected = (activeNetwork != null );

`     if (isConnected) {

              Toast.makeText(context, "Network is connected",
                       Toast.LENGTH_LONG).show();

       } else {

             Toast.makeText(context, "Network connection is
                   changed ",Toast.LENGTH_LONG).show();

          }

    }

 }
```

CHAPITRE 6

Les bases de données SQLite

Objectifs unitaires de l'enseignement :

- Apprendre à faire persister les données de l'application Android à l'aide des bases de données SQLite.
- Apprendre comment permettre le partage de données inter-applications mobiles Android.

1. Aperçu géneral

Une application mobile Android nécessite dans certains cas de stocker des données localement. Par exemple une application mobile e-commerce a besoin de stocker les informations concernant les différents produits à exposer aux clients dans une base de données locale. L'une des méthodes les plus efficaces pour assurer la persistance des données de l'application mobile est bel et bien l'utilistaion des bases de données. L'avantage de l'utilisation d'une base de données est qu'elle permette de manipuler et de stocker de façon flexible des données complexes et structurées.

Android fournit un support de bases de données relationnelles au travers de SQLite. En effet, SQLite est une base de données relationnelle légère, gratuite et Open Source ce qui a favorisé son adoption par les applications mobiles Android.

Du fait que les applications mobiles sont destinées à opérer sur des appareils mobiles ayant peu d'espace de stockage, de mémoire vive et de puissance de traitement. Il est donc déconseillé de mettre des volumes importants de données dans la base de données de l'application ou même de l'interroger assez fréquemment.

L'API SQLite est intégrée dans chaque appareil Android et pour l'utilisation d'une base de données SQLite sous Android on doit uniquement définir les instructions SQL pour créer et mettre à jour la base de données. Ensuite, celle-ci est gérée automatiquement par la plate-forme Android.

La figure ci-dessous présente les classes de l'API SQLite sous Android et le role de chacune d'elle.

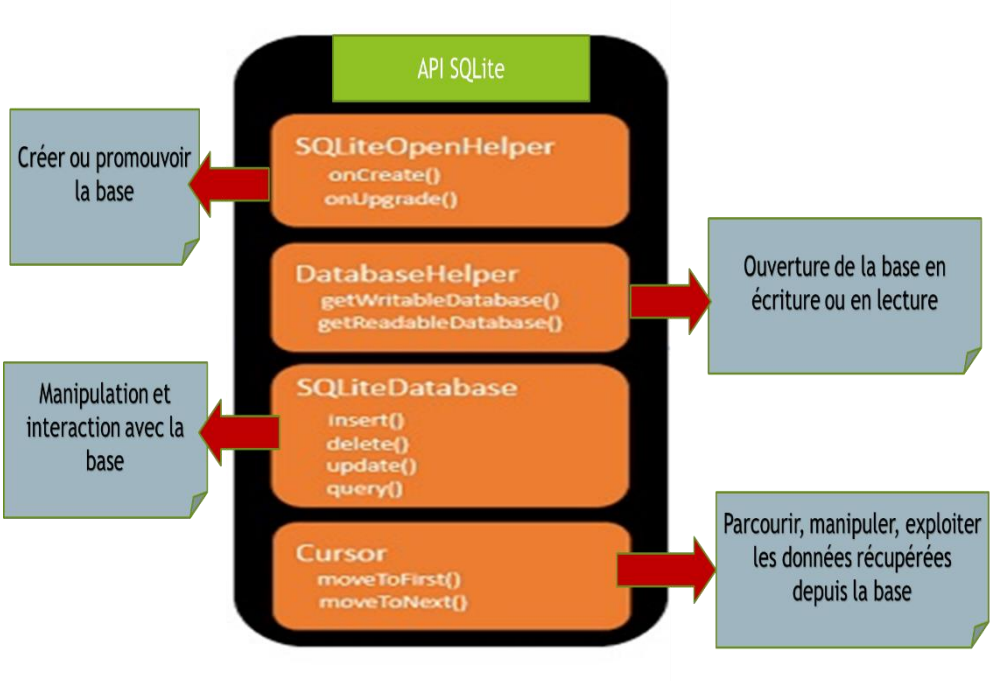


Figure 6.1. API SQLite sous Android.

2. Création de la base de données SQLite

La classe `SQLiteOpenHelper` est une classe qui permet la création et la gestion d'une base de données SQLite. Donc il est nécessaire de créer une classe qui hérite de la classe SQLiteOpenHelper et surcharger les méthodes onCreate() et onUpgrade()qui définissent le cycle de vie de l'instance en question :

- **onCreate()** lors de la création de l'instance la première fois,.

  Signature : `public void onCreate (SQLiteDatabase db);`

- **onUpgrade()** est appelée quand la version actuelle de la base est supérieure à la dernière version (migration d'une version ancienne à une nouvelle version).

  Signature : `public void onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion);`

❖ **Exemple :**

L'exemple ci-dessous illustre la création d'une table comptes (accounts) au profit de la base de données banque (bank).

```
public class MySQLiteOpenHelper extends SQLiteOpenHelper {

    // Nom de la base de données SQLite
```

```
private static final String DATABASE_NAME = “BANK.db";
// Nom de la table
private static final String TABLE_NAME = "accounts";
// Version de la base de données
private static final int DATABASE_VERSION = 1;
// Les noms des colonnes de la table
private static final String ID = "_id";
private static final String Name = "name";
private static final String Amount = "amount";
// Requete SQL pour la création de la table
private static final String CREATE_TABLE =
"CREATE TABLE TABLE_NAME ( ID INTEGER PRIMARY KEY
     AUTOINCREMENT,Name TEXT,   Amount REAL );’’;
// Requete SQL pour supprimer la table
private static final String DROP_TABLE ="DROP TABLE IF EXISTS"
     + “accounts”;
// Constructeur de la sous classe MySQLiteOpenHelper qui hérite de SQLiteOpenHelper
public MySQLiteOpenHelper(Context context, String DBname,
     CursorFactory factory,  int DBversion) {
     super(context, DBname, factory, DBversion);
     // Factory est généralement null }
@Override
public void onCreate(SQLiteDatabase db) {
     db.execSQL(CREATE_TABLE);
     }
@Override
```

```
    public void onUpgrade(SQLiteDatabase db, int oldVersion,
        int newVersion) {
        db.execSQL (DROP_TABLE);
        onCreate(db);
         }
} // Fin de la classe MySQLiteOpenHelper
```

3. Ouverture de la base de données SQLite

Pour l'ouverture de la base de données, il faut appeler les méthodes `getWritableDatabase()` ou `getReadableDatabase()` de la classe SQLiteOpenHelper dans l'activité de l'application.

Systématiquement, `getWritableDatabase()` et `getReadableDatabase()` créent (si elle n'existe pas) et ouvrent une base de données, puis renvoient l'objet de base de données qui peut être utilisé pour l'écriture ou lecture.

❖ **Exemple**

```
MySQLiteOpenHelper mySQLiteOpenHelper = new
   MySQLiteOpenHelper(getbaseContext(), "BANK.db", null, 1);
SQLiteDatabase db = mySQLiteOpenHelper.getWritableDatabase();
// Ou: SQLiteDatabase db = mySQLiteOpenHelper.getReadableDatabase();
```

4. Manipulation de la base de données SQLite

SQLiteDatabase est la classe de base pour manipuler une base de données SQLite sous Android. Elle fournit des méthodes pour ouvrir, effectuer des requêtes, mettre à jour et fermer la base de données.

SQLiteDatabase fournit plusieurs méthodes pour la manipulation de la base de données SQLite, tel que : **`query ()`, `rawQuery ()`, `insert()`, `update()`, `delete()`,** etc.

Elle fournit également la méthode qui permet d'exécuter une requuete SQL directement.

```
void execSQL(String sql, Object[] bindArgs);
```

Ou:

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

Les signatures ainsi que l'utilité de chacune des méthodes de manipulation de la base de données SQLite :

- **`Cursor query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit);`**

```
//Méthode pour interroger la table donnée et renvoyer un curseur sur l'ensemble de
//résultats.
```

- **`Cursor rawQuery (String sql, String[] selectionArgs);`**

```
//Exécute la requête SQL SELECT et renvoie un curseur sur l'ensemble de résultats.
```

- **`long insert (String table, String nullColumnHack, ContentValues values);`**

```
// Méthode pour insérer des lignes dans la table de la base de données.
```

- **`int update (String table, ContentValues values, String whereClause, String[] whereArgs);`**

```
// Méthode pour modifier des lignes dans la table de la base de données.
```

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

```
// Méthode pour supprimer des lignes dans la table de la base de données.
```

4.1. L'insertion

On distingue deux méthodes possibles de la classe SQLiteDatabase qui permettent l'insertion de données dans les table de la base de données. Il s'agit bien de la méthode : **`execSQL()`** ou **`insert()`.**

4.1.1. Insertion avec execSQL ()

```
void execSQL(String sql, Object[] bindArgs);
```

Tel que: le premier paramètre est une requete SQL INSERT INTO et le deuxième paramètre est ses arguments, comme indiqué dans l'exemple suivant :

❖ **Exemple**

```
db.execSQL("INSERT INTO accounts(name, amount) VALUES(?, ?)", new
Object[] {"Ahmed", 3000});
```

```
//ou db.execSQL("INSERT INTO accounts(name, amount) VALUES(" Jack", "3000"); ");
```

4.1.2. Insertion avec insert () et l'objet ContentValues

```
long insert (String table, String nullColumnHack, ContentValues
                                                        values);
```

Dans ce cas, il faut d'abord créer un objet de la classe ContentValues et lui associer les données via la méthode :

```
void put (String clé, Integrer|Float|String... valeur);
```

Ensuite il suffirait de passer l'objet ContentValues comme paramètre de la méthode `insert()`.

❖ **Exemple**

```
// Créer objet values de la classe ContentValues

   ContentValues values = new ContentValues();

// Ajout des données à insérer via la méthode put ()

   values.put("name", ''Amine");

   values.put("amount", 4000);

// La méthode inert ()

   long id = db.insert("account", null, values);
```

4.2. L'intérrogation

La Base de données SQLite s'interroge à l'aide de la méthode `query()` ou `rawQuery()`

4.2.1. Interrogation avec query ()

```
Cursor query (String table, String[] columns, String selection,
String[] selectionArgs, String  groupBy, String having, String
orderBy, String limit);
```

- table: la table de la base de données
- columns: colonnes à retourner (null => toutes les colonnes).
- selection: les colonnes concernées par la clause WHERE
- selectionArgs: les values de la clause WHERE.
- groupBy: spécifier les colonnes groupe des lignes (si null donc pas de regroupement).
- having: filter de selection des lignes regroupées à inclure dans objet cursor
- orderBy: DESC pour un ordre déscendant ou ASC pour un ordre ascendant (si null donc utilizer l'order par défaut).
- limit: nombre limite de lignes à retourner (si null donc pas de limite).

❖ **Exemple 1**

```
// SELECT * FROM accounts ORDER BY amount DESC LIMIT 3

Cursor cursor = db.query("accounts", null, null, null, null,
                null, "amount DESC", "3");
```

❖ **Exemple 2**

```
// SELECT * FROM accounts GROUP BY name HAVING amount >= 5000

Cursor cursor = db.query ("accounts", null, null, null,
     "name", "amount >= 5000",null, null);
```

❖ **Exemple 3**

```
// SELECT  name, amount FROM accounts WHERE name= Sahraoui AND
//amount >= 5000

Cursor cursor = db.query ("accounts", new String[] { "name",
"amount" }, "name =? and   amount >=?", new String[] {
"Sahraoui", 5000}, null, null, null, null);
```

4.2.2. Interrogation avec rawQuery ()

La méthode `rawQuery()` accepte directement une requête SQL SELECT en entrée.

```
Cursor rawQuery (String sql, String[] selectionArgs);
```

❖ **Exemple**

```
Cursor cursor = db.rawQuery("SELECT name, amount FROM accounts
                WHERE name = ?", new String[]{"Sofiane"});
```

Récupérer des éléments issus du résultat d'une requête de sélection se fait au travers de l'objet Cursor retourné par la méthode `query()` ou `rawQuery()`.

Au lieu de retourner tous les résultats, le Cursor agit comme un curseur de base de données accessible directement depuis les API. De cette façon, on se déplace au sein des résultats en modifiant la position de l'objet Cursor, via les méthodes de la classe Cursor. La table ci-dessous en présente les méthodes les plus importantes.

Table 6.1. Les méthodes principales de la classe Cursor.

Méthode	Description
moveToFirst ()	Déplace le curseur à la première position pour lire les données de la première ligne.
moveToLast ()	Déplace le curseur à la dernière position pour lire les données de la dernière ligne de la requête
moveToNext ()	Déplace le curseur d'une position pour lire la ligne suivante.
moveToPrevious ()	Déplace le curseur d'une position pour lire la ligne précédente.
moveToPosition(int)	Déplace le curseur à la position indiquée.
isFirst ()	Revoie vrai ou faux suivant que le curseur pointe vers la première ligne ou non
isLast ()	Revoie vrai ou faux suivant que le curseur pointe vers la dernière ligne ou non
isAfterLast ()	Revoie Vrai ou faux suivant que le curseur pointe après la dernière ligne ou non
getCount ()	Retourne le nombre de lignes qui sont renvoyées par la requete
getColumnName (int columnindex)	Retourne le nom de la colonne spécifiée par son index
getFloat (int columnIndex)	Retourne la valeur comme Float
getInt (int columnIndex)	Retourne la valeur comme Int
getString (int columnIndex)	Retourne la valeur comme String
Autres méthodes sur :	https://developer.anroid.com/reference/android/database/Cursor

4.3. La mise à jour

La modification des données des tables de la base de données se fait avec la méthode `update()` et un objet ContentValues tout comme l'insertion.

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

❖ **Exemple**

```
ContentValues values = new ContentValues();

values.put("amount", 5000);

int rows = db.update("accounts", values, "name = ?", new
                              String[]{''Sahraoui"});

String msg = "Totoally " + rows + " updated!";

Toast.makeText(this, msg, Toast.LENGTH_LONG).show();
```

4.4. La suppression

Il est possible de supprimer des données depuis la base de données à l'aide de la méthode `delete()`.

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

❖ **Exemple**

```
int rows = db.delete("accounts", "name = ?", new
                                        String[]{"Asma"});

String     msg = "Totoally " + rows + " rows deleted!";

Toast.makeText(this, msg, Toast.LENGTH_LONG).show();
```

La figure ci dessous résume les opérations de création et de manipulation de la base de données SQLite dans une application mobile Android.

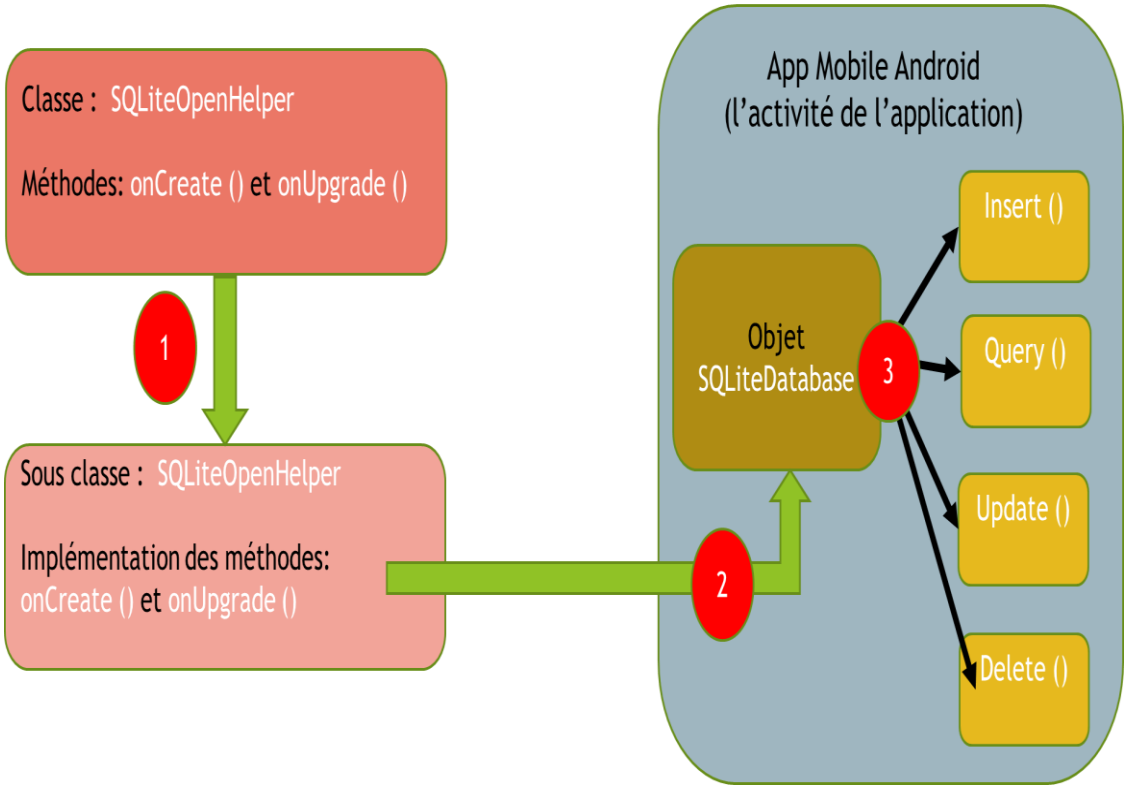


Figure 6.2. Etapes de création et de manipulation de base de données SQLite.

5. La fermeture de la base de données SQLite

Les opérations d'ouverture de la base de données (les méthodes `getWritableDatabase()` et getReadableDatabase()) sont assez couteuses. Par conséquent, on doit garder la base de données ouverte le plus long temps possible. La fermeture de la base de données SQLite est matérialisée par la méthode close()de la classe SQLiteDatabase.

La méthode `close()` devrait etre appelée depuis la méthode callback `onDestroy()` de l'activité manipulant la base de donnée.

❖ **Exemple**

```
protected void onDestroy() {
      super.onDestroy();
      db.close();
}
```

6. Le fournisseur de contenu

Les fournisseurs de contenu (en anglais : *Content Provier*) peuvent aider une application à gérer l'accès aux données stockées par elle-même, et fournir un moyen de partager des données avec d'autres applications. Ils encapsulent les données et fournissent des mécanismes pour définir leur sécurité (le service de contrôle d'accès).

En quelque sorte, les fournisseurs de contenu sont l'interface standard qui connecte les données dans un processus avec du code exécuté dans un autre processus. On peut toutefois configurer un fournisseur de contenu pour permettre à d'autres applications d'accéder, en toute sécurité, et modifier les données d'application.

Le content provider permet généralement de faire partager les données de la base de donnée SQLite de l'application avec d'autres applications. Si nous ne prévoyons pas de partager les données de notre Application, on peut ne pas utiliser ce composant car ce n'est pas obligatoire. Une fois le Content Provider est configuré, les applications externes devront (dans le but d'accéder aux données exposées) lui fournir une URI que ce dernier analysera pour retourner les données appropriées.

La figure suivante illustre l'objectif du fournisseur de contenu dans une application mobile.

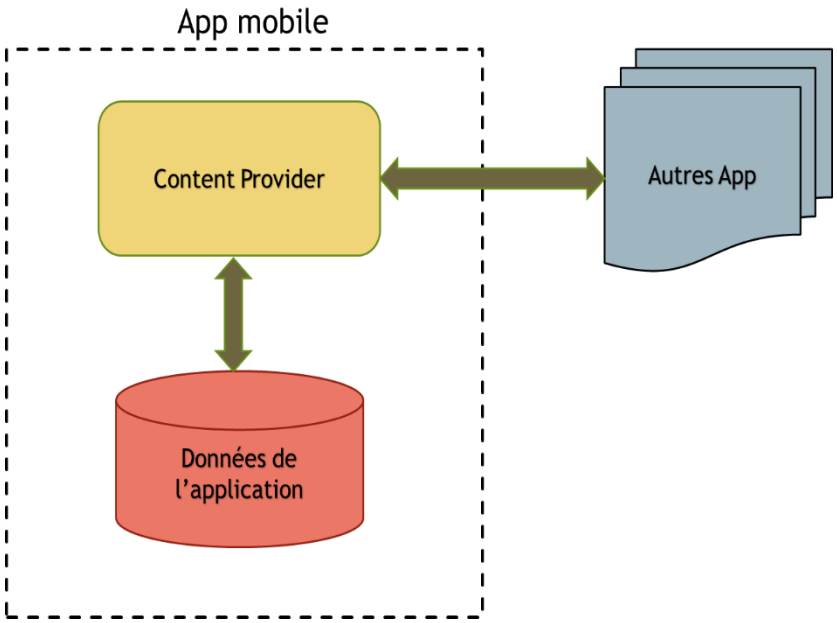


Figure 6.3. Objectif du fournisseur de contenu.

6.1. La création

Pour la création du fournisseur de contenu, il faut créer une sous classe héritant de la classe ContentProvider. Le but étant d'exposer les données d'une base de données SQLite.

La nouvelle sous classe héritant de ContentProvider doit absolument implémenter les six méthodes suivantes:

- **onCreate() :** Représente le point d'entrée du Content Provider. Vous pourrez donc y initialiser différentes variables qui vous serviront par la suite (exemple: instancier objet de la base de données SQLite).
- **query():** Cette méthode permettra de récupérer depuis le content provider les données dans un Cursor. cette méthode est utilisée quand le Content Provider est appelé pour récupérer des données depuis l'endroit spécifié par URI (la base de donnée). Voici un exemple d'URI:

```
public static final Uri CONTENT_URI =
      Uri.parse("content://com.MyDataBases.ExempleDB/DB_TABLE);
```

- **insert():** Cette méthode permet d'insérer des données (ex: une ligne de table) au format ContentValues dans la destination indiquée par URI (une table de BD SQLite).
- **delete():** Cette méthode permet de supprimer une ou plusieurs lignes dans la table de DB SQLite indiquée par URI.
- **update():** Cette méthode permet de modifier une ou plusieurs lignes dans la table de DB SQLite indiquée par URI.
- **getType():** permet de retourner le type MIME associé à l'URI permettant d'identifier plus précisément le type des données qui seront retournées.

❖ Prototype de création

```
public class ExempleContentProvider extends ContentProvider {

    @Override

    public boolean onCreate () {

        return true/false;

    }

    @Override
```

```
public Cursor query (Uri uri, String[] projection, String
        selection, String[] selectionArgs, String sortOrder)
    {
    ..................
    }
@Override
public String getType (Uri uri) {
     ..................
   }
@Override
public Uri insert (Uri uri, ContentValues contentValues) {
        ..................
}
@Override
public int delete (Uri uri, String selection, String[]
    selectionArgs) {
          ..................
}
@Override
public int update (Uri uri, ContentValues contentValues,
    String selection, String[] selectionArgs) {
          ..................
  }
}
```

6.2. La déclaration dans le fichier AndroidManifest.xml

La déclaration du fournisseur de contenu ans le manifest de l'application se fait à l'aide du tag `<provider>`. Au sein du tag provider on spécifie :

- `android:name` : le nom du fournisseur.
- `android:authorities` : l'autorité du fournisseur.
- `android:enabled` : activer ou désactiver le fournisseur
- `android:exported` : permettre l'accès au fournisseur epuis une autre application ou non.

```
============================================================
   <?xml version="1.0" encoding="utf-8"?>
   <manifest
       xmlns:android="http://schemas.android.com/apk/res/android"
       package="com.openclassrooms.savemytrip">
       ...
       <application ...>
           ...
           <provider
             android:name=".provider.ExempleContentProvider"
             android:authorities="com.exemple.fournisseur”
             android:enabled="true"
             android:exported="true"/>
        </application>
   </manifest>
============================================================
```

Tous les fournisseurs de contenu doivent avoir associé à eux une autorité et un content URI. L'autorité est généralement le nom complet du package de la classe de fournisseur de contenu elle-même, tel que déclaré lorsque la nouvelle classe de fournisseur de contenu a été créée. En effet, l'URI du contenu varie en fonction des exigences de l'application, il comprendra l'autorité avec le nom de la base de données (ou directement la table) à manipuler via le content provider. L’URI du contenu c’est donc l’identificateur ou schéma d’accès aux données que l’on veuille modifier ou partager dans le content provider.

CHAPITRE 7

Les services

Objectifs unitaires de l'enseignement :

- Découvrir et cerner le rôle des services dans les applications mobiles Android.
- Apprendre comment créer et manipuler les services dans les applications mobiles Android.

1. Présentation générale des services

Le service est un composant de l'application mobile Android qui effectue des opérations de longue durée dont l'utilisateur est éventuellement averti. En effet, le service peut être démarré par un autre composant de l'application et il continue à s'exécuter en arrière-plan même si l'utilisateur passe à une autre application. De plus, un composant peut se lier à un service pour interagir avec lui et même effectuer une communication interprocessus (IPC) par exemple, un service peut gérer les téléchargements depuis internet, lire de la musique, effectuer des entrées/sorties sur des fichiers ou même interagir avec un fournisseur de contenu, le tout généralement en arrière-plan.

Il est important de signaler que le service ne fournit pas d'interface utilisateur et donc il n'est pas lié au cycle de vie de l'activité. D'un autre part, et de point de vue exécution du service, on distingue deux catégories de services : les services qui s'exécutent en en premier plan (*foreground*) et les services qui s'exécutent en arrière-plan (*background*).

- **Les services du premier plan**

Un service qui s'exécute en premier plan effectue une opération qui est visible par l'utilisateur. Par exemple, une application audio utiliserait un service de premier plan pour lire une piste audio. Il faut noter que les services du premier plan doivent obligatoirement afficher une notification à l'utilisateur. Ainsi, les services de cette catégorie continuent à fonctionner même lorsque l'utilisateur n'interagit pas avec l'application.

- **Les services d'arrière-plan**

Un service d'arrière-plan effectue une opération qui n'est pas directement remarquée par l'utilisateur par exemple, si une application utilise un service pour compacter ses données du stockage, il s'agit généralement d'un service d'arrière-plan.

❖ **Remarque**

Un service (s'exécutant en premier ou en arrière-plan) est toujours plus prioritaire qu'une activité invisible. Cependant, un service qui s'exécute en arrière-plan est moins prioritaire qu'une activité visible et le service qui s'exécute en premier plan peut avoir la même priorité que les activités du premier plan. Mais, dans ce cas, il est nécessaire d'avoir une notification visible pour le service associé. Les services du premier plan sont fréquemment utilisés pour gérer la lecture des vidéos ou de la musique

Par défaut, un service s'exécute dans le même processus que le thread principal de l'application. Un modèle couramment utilisé pour une implémentation de service consiste

à créer et à exécuter un nouveau thread dans le service pour effectuer le traitement en arrière-plan, puis à terminer le service une fois le traitement est terminé.

2. Les types de services

De point de vue appartenance ou non appartenance du service au contexte de l'application, on distingue deux types de services :

2.1. Service démarré (*started service*)

Également appelé ''service local'' Ce service est créé lorsqu'un autre composant appelle la méthode `startService()`. Le service s'exécute indéfiniment et doit s'arrêter en appelant `stopSelf()`. Notons qu'un autre composant puisse également arrêter le service en évoquant la méthode `stopService()`. Finalement, lorsque le service est arrêté, le système le détruit tout simplement. La figure ci-dessous illustre le mode de fonctionnement des services démarrés.

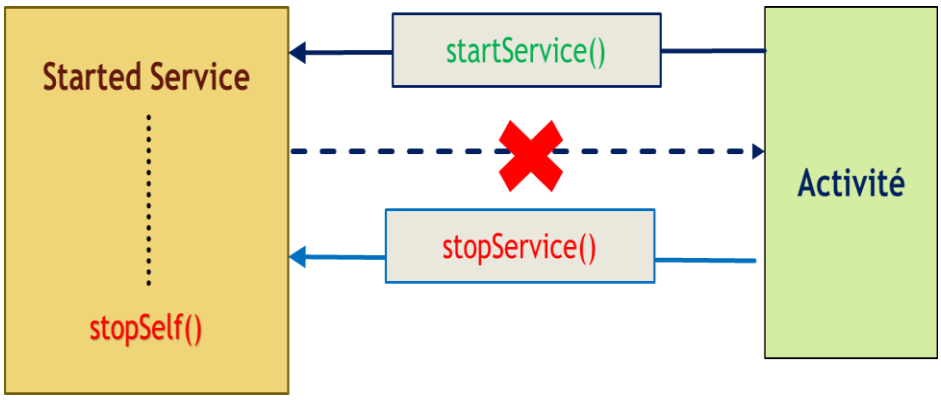


Figure 7.1. Principe du service démarré.

2.2. Service lié (*Bound service*)

Egalement appelé ''service distant''. Tel type de services est créé lorsqu'un autre composant (nommé le client) appelle `bindService()`. Le client communique ensuite avec le service via une interface `Ibinder` et il peut fermer la connexion avec le service en appelant `unbindService()`.

Plusieurs clients peuvent se lier au même service et lorsqu'ils en se délient tous, c'est à ce niveau-là que le système détruise le service. La figure en bas montre le modèle de fonctionnement des services liés.

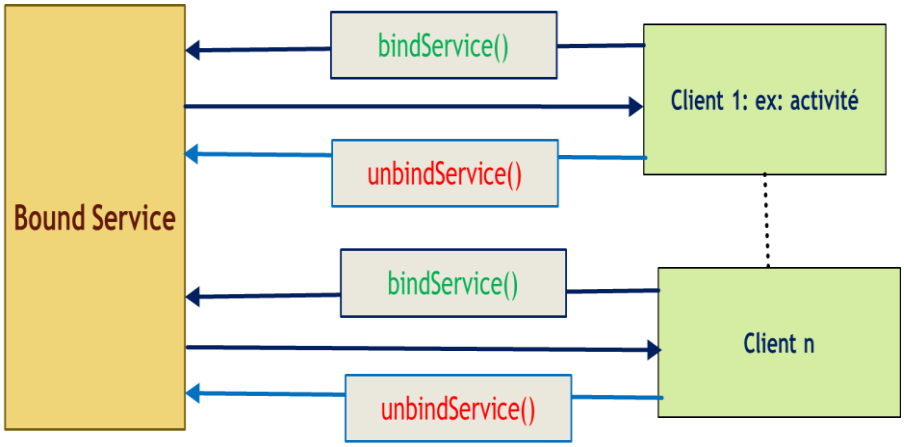


Figure 7.2. Principe du service lié.

❖ Remarques

- Il est possible de se lier à un service déjà démarré avec `startService()`
- Par exemple, à partir d'une activité, on peut démarrer un service pour la lecture de la musique en appelant `startService()` avec un intent qui identifie la musique à jouer. Plus tard, lorsque l'utilisateur souhaite exercer un certain contrôle sur le lecteur ou obtenir des informations sur le morceau en cours, l'activité peut se lier au service en appelant `bindService()`. Dans de tels cas, `stopService()` ou `stopSelf()` n'arrête pas réellement le service jusqu'à ce que tous les clients s'en délient.

3. Les cycles de vie des services

Le service démarré se lance lorsqu'un composant évoque la méthode `startService()`. A ce niveau, le service exécute les méthodes callback `onCreate()` et `onStartCommand()` et réalise les actions qui lui sont affectées. Le service est arrêté à la demande du composant qui l'a démarré ou il arrête lui-même (comme mentionné dans la section précédente). Dans ce cas, le service exécute la méthode `onDestroy()`.

Quant au service lié, le cycle de vie commence plutôt quand le client appelle la méthode `bindService()`. Le service est démarré et exécute la méthode callback `onBind()` après avoir exécuté la méthode `onCreate()`. Le service effectue sa tâche et il s'arrête, en appelant `onDestroy()`, une fois que tous les clients qui lui sont connectés s'en détachent en appelant la méthode `unbindService()`. Il est toutefois possible pour un client qui vient juste de se détacher du service (la méthode `onUnbind()` est en cours d'exécution), de se relier au même service en appelant `bindService()` à nouveau. La figure ci-dessous illustre les cycles de vie des eux types de services.

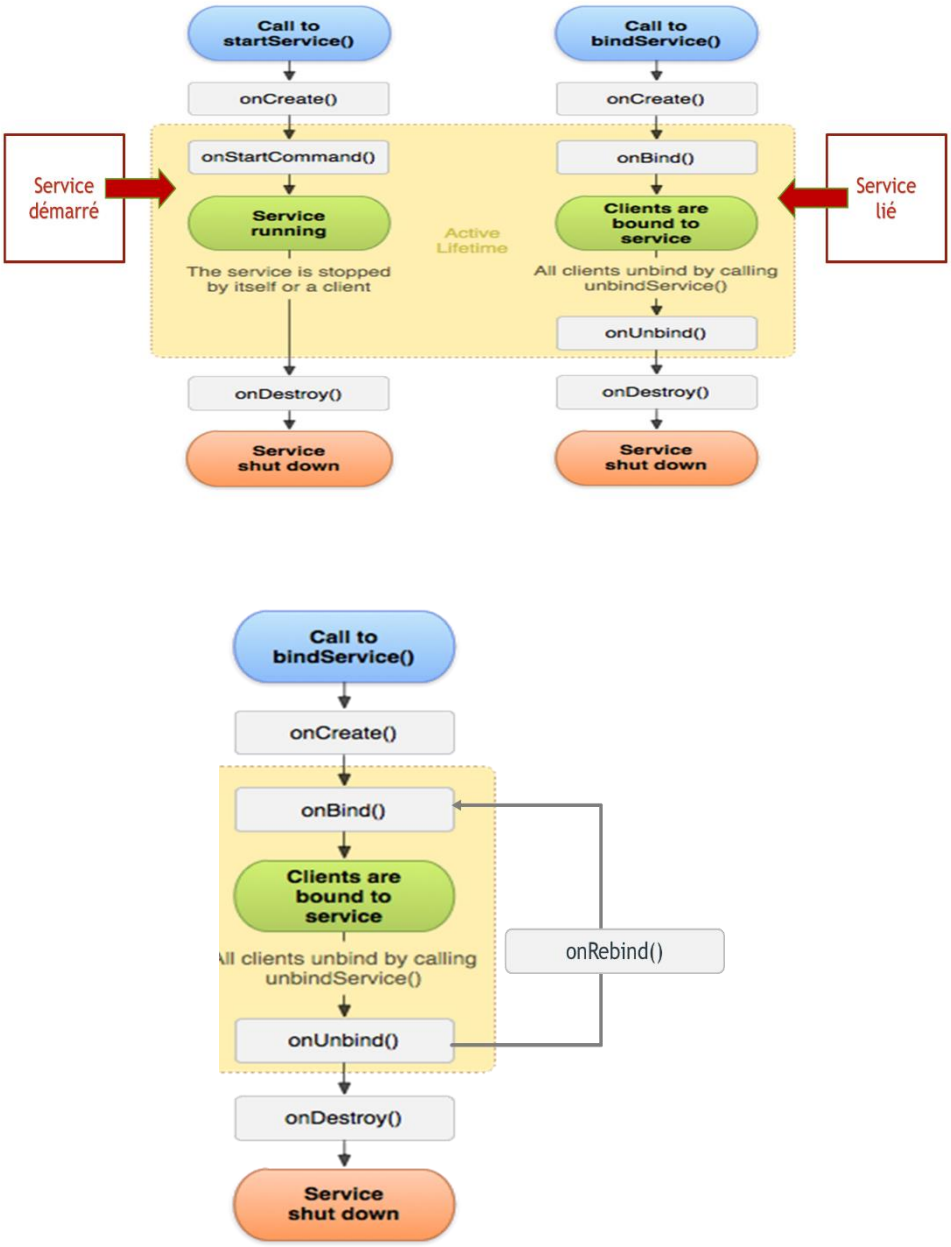


Figure 7.3. Cycles de vie des services démarrés et liés.

Le tableau ci-dessous résume la description des méthodes du cycle de vie des services

Table 7.1. Description des méthodes callback du cycle de vie du service.

Méthode callback	Description
onCreate ()	Le système appelle cette méthode lorsque le service est créé pour la première fois.
onStartCommand ()	Le système appelle cette méthode lorsqu'un autre composant, tel qu'une activité, demande le démarrage du service, en appelant startService (). Si vous implémentez cette méthode, il est de votre responsabilité d'arrêter le service lorsque son travail est terminé, en appelant les méthodes stopSelf () ou stopService ().
onBind ()	Le système appelle cette méthode lorsqu'un autre composant souhaite se lier au service en appelant bindService (). Si vous implémentez cette méthode, vous devez fournir une interface que les clients utilisent pour communiquer avec le service, en renvoyant un objet IBinder. Vous devez toujours implémenter cette méthode, mais si vous ne souhaitez pas autoriser la liaison, vous devez retourner null.
onUnbind ()	Le système appelle cette méthode lorsque tous les clients se sont déconnectés d'une interface particulière publiée par le service.
onRebind ()	Le système appelle cette méthode lorsque de nouveaux clients se sont connectés au service, après avoir été préalablement informé que tous s'étaient déconnectés via onUnbind (Intent).
onDestroy ()	Le système appelle cette méthode lorsque le service n'est plus utilisé et est en cours de destruction. Votre service doit implémenter cela pour nettoyer toutes les ressources telles que les threads, les écouteurs enregistrés, les récepteurs, etc.

4. La déclaration du service

Un service doit être déclaré dans le fichier *AndroidManifest.xml* via le tag `<service>` où :

- l'attribut `android:enabled` sert pour activer ou désactiver le service
- l'attribut `android:exported` spécifie si d'autres composants puissent se connecter au service.

```
<service
        android:name="MyService"
        android:enabled="true"
        android:exported="true" >
</service>
```

5. La création du service

Pour la création d'un service (lié ou démarré), il suffit de créer une classe qui hérite de la classe `Service` et implémenter les méthodes du cycle de vie comme montré dans le code ci-dessous.

```
============================================================
   public class MyService extends Service {
            int startMode;
            IBinder binder;
            boolean allowRebind;
            @Override
            public void onCreate () {
               // Le service est créé
            }
            @Override
            public int onStartCommand (Intent intent, int flags,
              int startId) {
               // Le service est démarré à cause de l'appel à la méthode startService()
                        return startMode;
            }
            @Override
            public IBinder onBind (Intent intent) {
                  // un client est lié au service avec la méthode bindService()
                        return binder;
                    //return null si on veut interdire la liaison avec le service.
            }
            @Override
            public boolean onUnbind (Intent intent) {
              // Tous les clients de délient du service avec la méthode unbindService()
                        return allowRebind;
            }
        @Override
            public void onRebind (Intent intent) {
               // un client est relie au service avec la méthode bindService(), après que
              // la méthode onUnbind() a été déjà  appelée.
            }
            @Override
```

```
        public void onDestroy () {
          // Le service est non plus utilisé et il vient d'être détruit
              }
    }
```

==========

6. Démarrer un service

Un composant de l'application Android (exemple : activité ou *Broadcast Receiver*) démarre le service via la méthode `startService();`

- **Exemple :**

```
Intent I = new Intent(this, MyService.class);

I.putExtra ("KEY", Valeur qui va être utilisée par le service);

this.startService(I);
```

Alternativement, un composant peut démarrer et se lier à un service via l'appel de méthode `bindService()`. Cela permet de communiquer directement avec le service.

Si la méthode `startService()` est appelée et le service n'est pas encore en cours d'exécution. L'objet service est créé et la méthode `onCreate()` du service est appelée.

Une fois le service est démarré, la méthode `onStartCommand()` du service est appelée et elle transmet l'objet Intent à partir de l'appel `startService()`.

Si `startService()` est appelé pendant que le service est en cours d'exécution `onStartCommand()` est également appelé. Par conséquent, le service doit être préparé pour que `onStartCommand()` puisse être appelée plusieurs fois.

Notons bien que le service n'est démarré qu'une seule fois, quelle que soit la fréquence à laquelle nous appelons la méthode `startService()`.

7. Redémarrage du service

Dans son appel à la méthode `onStartCommand()`, le service renvoie un entier qui définit son comportement de redémarrage au cas où le service serait arrêté par le système Android.

Les options les plus courantes sont :

- `Service.START_STICKY`
- `Service.START_NOT_STICKY`
- `Service.START_REDELIVER_INTENT`

Les trois options sont décrites dans le tableau ci-dessous.

Table 7.2. Options de redémarrage du service.

Option	Description
Service.START_STICKY	Le service est redémarré s'il est interrompu. Les données de l'intent sont transmises à la méthode onStartCommand () sont nulles. Utilisée pour les services qui gèrent leur propre état et ne dépendent pas des données de l'intent
Service.START_NOT_STICKY	Le service n'est pas redémarré. Utilisé pour les services qui sont de toute façon périodiquement déclenchés. Utilisé pour indiquer à l'OS de ne pas déranger et recréer le service
Service.START_REDELIVER_INTENT	Similaire à Service.START_STICKY mais l'intent d'origine est à nouveau passé à la méthode onStartCommand ().

8. Arrêter un service

On arrête un service démarré avec la méthode `stopService()`. Quelle que soit la fréquence de l'appel à la méthode `startService()`, un appel à la méthode `stopService()` provoque l'arrêt immédiat du service.

Un service peut terminer lui-même en appelant la méthode `stopSelf()`. Cela est généralement le cas quand le service termine son travail.

Le cas échéant, le service lié ne peut s'arrêter qu'après la déconnexion de tous les clients qui lui sont liés. La déconnexion des clients est explicitée avec l'appel à la méthode `unbindservice()`.

9. La communication avec le service

Deux méthodes possibles pour implémenter la communication entre un service et une activité :

9.1. Utilisation des Intents

Dans un scénario simple, aucune communication directe n'est requise. Le service reçoit les données de l'intent du composant Android qui l'a lancé et effectue son travail. Aucune notification n'est nécessaire. Par exemple, si le service met à jour un fournisseur de contenu, l'activité est notifiée par le fournisseur de contenu et aucune étape supplémentaire du service n'est nécessaire. Cette approche fonctionne pour les services démarrés (locaux) et fonctionnant dans leur propre processus.

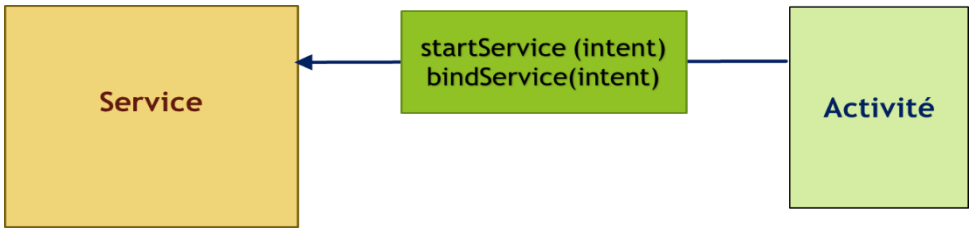


Figure 7.4. Communication avec le service en utilisant l'intent.

9.2. Utilisation du Broadcast Receiver

Il est possible de se baser sur des récepteurs enregistrés pour la communication avec le service. Par exemple, une activité peut enregistrer un récepteur de diffusion pour un événement et le service envoie les événements correspondants. Il s'agit d'un scénario très typique, dans lequel le service doit signaler à l'activité que son traitement est terminé.

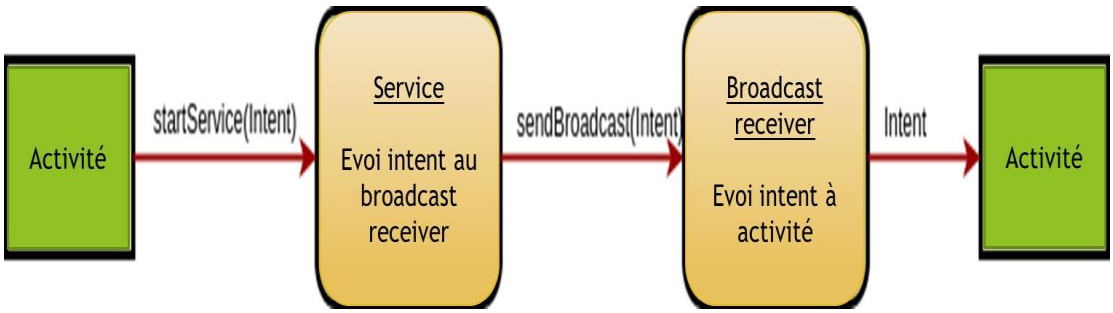
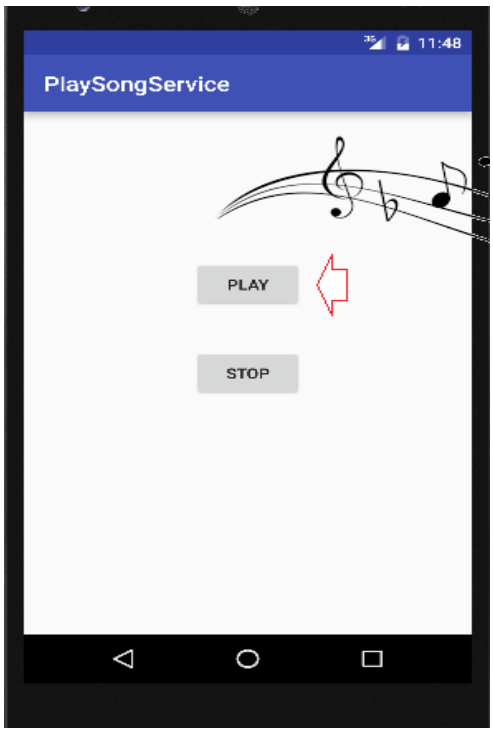


Figure 7.5. Communication avec le service à travers le récepteur de diffusion.

❖ Exemple

On a une Application **PlaySongService** dont l'activité principale présente deux boutons **Play** qui démarre un service de lecture de musique et **Stop** qui arrête le service.



- **Déclaration du service ans le AndroidManifest.xml de l'application :**

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.o7planning.playsongservice">

    <application ...>

        <service
            android:name=".PlaySongService"
            android:enabled="true"
            android:exported="true">
        </service>
            ....
    </application>
</manifest>
```

- **Code de la classe du service :**

```
public class PlaySongService extends Service {

       private MediaPlayer mediaPlayer;

       public PlaySongService() {       }

    @Override
     public IBinder onBind (Intent intent) {
    // Ce service n'est pas lié (unbounded) donc cette méthode ne sera pas appelée
                  return null;
       }


    @Override
      public void onCreate () {
               super.onCreate();
          // Créer objet MediaPlayer pour jouer le morceau musical
            mediaPlayer =
                MediaPlayer.create(getApplicationContext(),
                                                    R.raw.mysong);
       }


    @Override
        public int onStartCommand (Intent intent, int flags, int
                                                    startId) {
           // Jouer de la musique
                 mediaPlayer.start ();

               return Service.START_STICKY;
        }

    @Override
     public void onDestroy() {
           // Libérer les ressources
              mediaPlayer.release();

                  super.onDestroy();      }
}
```

- **Code de l'activité de l'application :**

```
public class MainActivity extends Activity {
      private Button buttonPlay;
      private Button buttonStop;

     @Override
     protected void onCreate(Bundle savedInstanceState) {

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

            buttonPlay = (Button)findViewById(R.id.button_play);
           buttonStop = (Button)findViewById(R.id.button_stop);

            buttonPlay.setOnClickListener(new View.OnClickListener()
            {

                  @Override
                        public void onClick(View v) {
                        playSong ();
                    }
                   });


            buttonStop.setOnClickListener(new View.OnClickListener()
            {

                  @Override
                      public void onClick (View v) {
                           stopSong ();
                  }
                   });
        }

        // Cette méthode est appelée quand utilisateur clique sur le bouton Play

public void playSong()  {

          // Créer objet Intent pour démarrer le service PlaySongService
            Intent myIntent = new Intent (this, PlaySongService.class);
         // Appeler la méthode startService () avec l'objet Intent comme paramètre
            this.startService (myIntent);      }

        // Cette méthode est appelée quand l'utilisateur clique sur le bouton Stop
```

```
    public void stopSong()  {

        Intent myIntent = new Intent (this, PlaySongService.class);
        this.stopService (myIntent);
    }

}
```

Collection d'exercices (avec et sans solutions)

Exercice 1 :

Donner le code d'une activité qui affiche son état dans le cycle de vie au user via un Toast.

Exercice 2 :

Complétez le scénario suivant avec les méthodes callback correspondantes au cycle de vie de l'activité, dans une application mobile Android

Sur un dispositif mobile doté du système Android, l'utilisateur lance l'application mobile YouTube et recherche une certaine vidéo à regarder. L'activité de l'application exécute **.....**

Après un bon moment, l'utilisateur navigue vers une autre application (Facebook par exemple) qui a été déjà lancée avant l'application YouTube. Dans ce cas, l'activité de l'application YouTube exécute **.....**et l'activité de l'application Facebook exécute **.....** En consultant les publications sur Facebook, l'utilisateur reçoit un appel téléphonique. L'activité de l'application Facebook exécute alors **.....**

Après la conversation téléphonique, l'utilisateur décide de fermer les activités des deux applications YouTube et Facebook. Ces activités exécutent **.....**

Exercice 3 :

Donner la représentation graphique des layouts suivant :

- **Layout 1 :**

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

  android:orientation="vertical"

  android:layout_width="fill_parent"

  android:layout_height="fill_parent" >

  <Button

    android:id="@+id/premier"

    android:layout_width="fill_parent"

    android:layout_height="wrap_content"

    android:text="Premier bouton" />
```

```
    <Button
    android:id="@+id/second"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Second bouton" />
</LinearLayout>
```

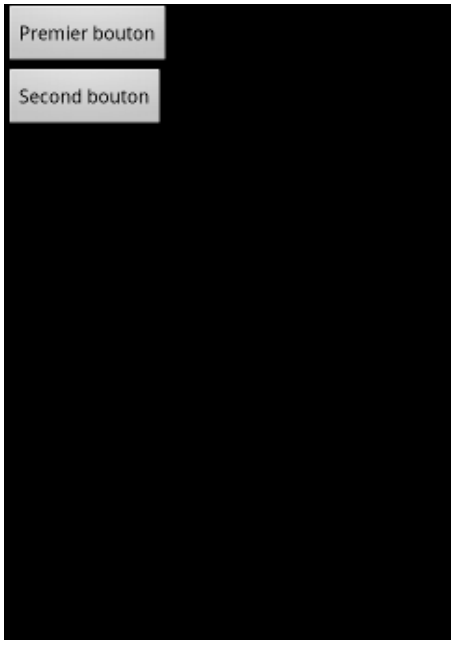
- **Layout 2 :**

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
      <Button
    android:id="@+id/premier"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Premier bouton" />
      <Button
    android:id="@+id/second"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Second bouton" />
</LinearLayout>
```

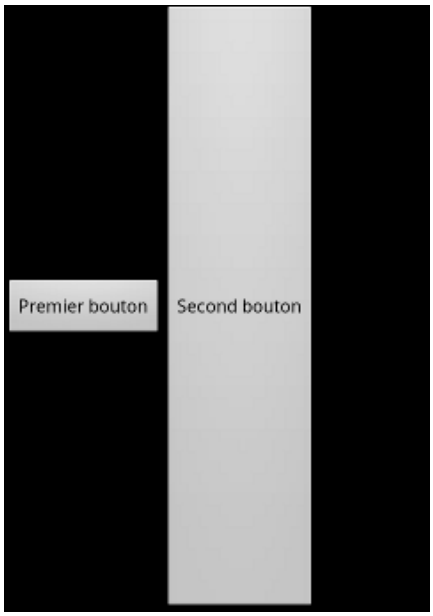
Exercice 4 :

Donner les layouts (en XML) correspondants aux interfaces suivantes :

- **Interface 1 :**



- **Interface 2 :**



Exercice 5 :

On souhaite développer une application Android permettant de saisir du texte dans un champ de saisie (editText1) qui sera envoyé à une autre activité (Activity2) lorsque l'utilisateur clique sur un bouton (button1).

N	MainActivity.java
1	`public class MainActivity extends Activity {`
2	`  @Override`
3	`  public void onCreate(Bundle savedInstanceState) {`
4	
5	`  setContentView(layout.activity_main);`

```
6
7           Button b = (Button) findViewById(R.id.button1);
8           b.setOnClickListener(new View.OnClickListener() {
9               public void onClick() {
10                  lancer ();
11              }
12          });
13    }
14
15    public void lancer() {
16          Intent i = new Intent(this,Activity2);
17          EditText e = (EditText) findViewById(R.id.editText1);
18          String c = e.getText().toString();
19          i.put("text", c);
20          startActivity(Intent);
21    }
22  }
```

- Le code ci-dessus contient **6 erreurs**. Pour chacune d'entre elles : indiquez son numéro de ligne dans le code, expliquez (très brièvement) son origine, et procéder à sa correction.

Supposons maintenant que le texte saisi dans editText1 est un lien (par exemple : https://univ-biskra;dz) et que l'on veut afficher le contenu web correspondant, lorsque le bouton est cliqué.

- Modifier la méthode lancer() pour répondre à ce besoin en utilisant l'Intent implicite qui convient.

Exercice 6 :

Le code ci-dessous comprend **5 problèmes**. Identifier et corriger chaque problème dans le tableau ci-dessous.

```
public class MainActivity extends Activity  {
   Button b1; EditText ed1;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      ed1 = (EditText)findViewById(R.editText);
      b1 = (Button)findViewById(R.id.button);
      b1.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick() {
```

```
            if(ed1.toString().equals("admin")) {
                  startActivity (I);
               }
         } });
   }
}
```

Problème	Correction

Exercice 7

Soit un layout linéaire qui dispose les objets graphiques suivant verticalement.

-Deux EditText : ET1 et ET2.

- Un boutton : Badd

-Un dernier TextView : TVRes

On veut que lorsqu'on clique sur le bouton Badd, le résultat d'addition des deux valeurs de ET1 et ET2 et les afficher dans TVRes.

Exercice 8

On veut que lorsqu'on clique sur le bouton Badd, le résultat s'affiche sur le TextView TVRes dans une deuxième activité.

- Supposons que la deuxième activité contienne en plus, un bouton Back pour revenir à l'activité principale (la première activité).

-Supposons cette fois-ci que la deuxième activité contient le TextView pour l'affichage du résultat, le bouton Back pour revenir en arrière et un bouton display pour l'affichage du contenu d'une page web dont le lien est ''https://www.android.com''. Modifier le code de l'activité pour supporter de telles modifications.

Exercice 9

Compléter le code de la méthode sendSMS() qui permet d'envoyer un SMS, dont le message est: ''**Good luck in your exam''** et le numéro du destinataire est : **12346556.**

```
public void sendSMS() {
      Uri numero = ..............("smsto:12346556");
      Intent I = new Intent (Intent.ACTION_SENDTO, numero);

       ..................................................
       ..................................................
}
```

Exercice 10

On a une application mobile pour la location de voiture avec envoi du choix de la voiture par SMS, la base de données LOCVOITURE contient plusieurs tables parmi elles la table voiture qui contient les champs suivants :

- ID : de type entier auto incrémenté et représente la clé primaire
- Marque : de type texte
- Type : de type texte
- nbPlaces : de type entier
- AnnéeCirculation : de type entier

1. La classe **DBLocation** est une classe qui hérite d'une classe connue et qui permet de créer la table. Donner la première ligne de cette classe et le code de la méthode onCreate()
2. Pour utiliser la table, on doit créer une instance de la classe **DBLocation** et ouvrir la table en mode écriture ; donner les deux lignes du code permettant l'instanciation et l'ouverture de la table
3. Donner l'instruction permettant d'insérer une voiture avec les valeurs suivantes **(''Renault'' , ''symbole'', 5, 2016)** en utilisant l'instruction INSERT INTO de SQL
4. Donner l'instruction permettant d'insérer une voiture avec les valeurs suivantes (''Renault'', ''symbole'', 5, 2016)**.** Utiliser **ContentValues**.
5. Donner l'instruction qui permet de supprimer toutes les voitures dont l'année de circulation est 2013.
6. Que fait le code suivant ?

```
Cursor cursor = db.query("voiture", String[]{"ID",
"Marque", "Type", "nbPlaces", "AnneeCirculation"}, null,
null, null, null, null, null);

cursor.moveToFirst()
While (!cursor.isAfterLast()) {
      int id = cursor.getInt(0);
       String marque= cursor.getString(1);
       String type = cursor.getString(2);
       int NBPlaces = cursor.getInt(3);
       int AnneeCirculation = cursor.getInt(4);

   String S = String.valueOf(id)+" "+marque+" "+type+" "+
   String.valueOf(NBPlaces)+" "+
    String.valueOf(AnneeCirculation)
    Log.i("ENI", S);
    cursor.moveToNext()
}
```

Exercice 11

Compléter le code suivant :

```
public class DataBaseFilm extends ……. {
      public DataBaseFilm(……, String nomBD, CursorFactory
      fact, int version)
      { ……(ctx, nomBD, fact, version); }
      @Override
      public void onCreate(……) {
      …… requete="CREATE TABLE tab_film (id INTEGER PRIMARY
      KEY AUTOINCREMENT, categorie TEXT, titre TEXT, realisateur
      TEXT);";
      db……..(requete);
      }
      @Override
      public void onUpgrade(SQLiteDatabase db, int arg1, int arg2)
      {
      db.execSQL("DROP TABLE IF IXISTS tab_film;");
      ……;
```

```
        }
}

public class MainActivity extends …… {

public …… onCreate(Bundle SavedInstanceState) {

DBFilm= new DataBaseFilm(getBaseContext(), "db_film.db", null, 1);
db=DBFilm.get…….();
ArrayList a=new ArrayListe();
String query="Select * from tab_film;";
…… res=db.rawQuery(query,null);
res.moveToFirst();
while (!res.isafterlast())
{               a.add(res…….(0)+"    "+res.getstring(1)+"
"+res.getstring(2)+" "+res.getstring(3));
res.movetoNext();
}
}
```

Exercice 12

Modifier le code de l'application **PlaySongService** vue dans l'exmple du chapitre 7 (les services) afin d'utiliser un service lié au lieu d'un service démarré.

Exercice 13

Choisissez une application mobile Android en vedette dans n'importe quel domaine (sport, musique, éducation, e-health, e-commerce, ...)

-En se basant sur les connaissances acquises dans le module, essayez de décortiquer l'application choisie. En d'autres termes, il vous est demandé d'analyser les différentes parties de l'application mobile que vous avez déjà choisie suivant les connaissances techniques que vous avez eues sur le développement des applications mobiles.

- Organiser votre réponse comme suit :

  - Aperçu sur l'application (son nom, logo, objectif, etc.).
  - Analyse de l'application :
    - Les activités nécessaires (leur nombre, l'interface utilisateur de chacune, etc.).

    - Les intents implicites et explicites existent dans l'application ou non ? Expliquer un peu.
    - L'application nécessite-t-elle une base de données ou non ?
    - Le content provider est-il nécessaire dans l'application choisie ou non (l'application a-t-elle besoin de partager ses données avec d'autres applications ou non) ?
    - Y a-t-il des services dans l'application ?
- Les critiques (partie facultative à ignorer si vous n'avez pas de critiques pour l'application)
    - Est-ce que l'application peut être améliorée ? que proposez-vous pour cela ?

Solutions des exercices

Solution 1

```
Public class MyActivity extends Activity
{
      @Override
      Protected void onCreate (Bundle SavedInstanceState)
      {
      Super.onCreate (SavedInstanceState);
      Toast.makeText (this,"onCreate",Toast.LENGTH_SHORT).show();
      }
      @Override
      Protected void onPause ()
      {
      Super.onPause ();
      Toast.makeText (this,"onPause",Toast.LENGTH_SHORT).show();
      }
      @Override
      Protected void onStop ()
      {
      Super.onStop ();
      Toast.makeText (this,"onStop",Toast.LENGTH_SHORT).show();
      }
      @Override
      Protected void onDestroy ()
      {
      Super.onDestroy ();
      Toast.makeText (this,"onDestroy",Toast.LENGTH_SHORT).show();
      }
```

Solution 2

Complétez le scénario suivant avec les méthodes callback correspondantes au cycle de vie de l'activité, dans une application mobile Android

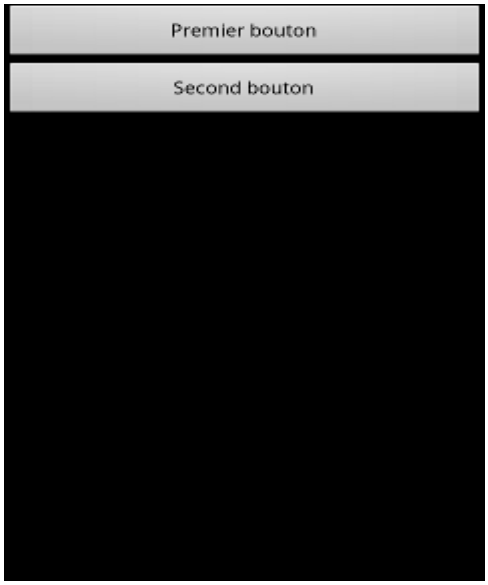
Sur un dispositif mobile doté du système Android, l'utilisateur lance l'application mobile YouTube et recherche une certaine vidéo à regarder. L'activité de l'application exécute **onCreate (), onStart() et onResume()**

Après un bon moment, l'utilisateur navigue vers une autre application (Facebook par exemple) qui a été déjà lancée avant l'application YouTube. Dans ce cas, l'activité de l'application YouTube exécute **onStop()** et l'activité de l'application Facebook exécute **onRestart() onStart() onResume()**. En consultant les publications sur Facebook, l'utilisateur reçoit un appel téléphonique. L'activité de l'application Facebook exécute alors **onStop() (ou onPause() ensuite onStop()).**

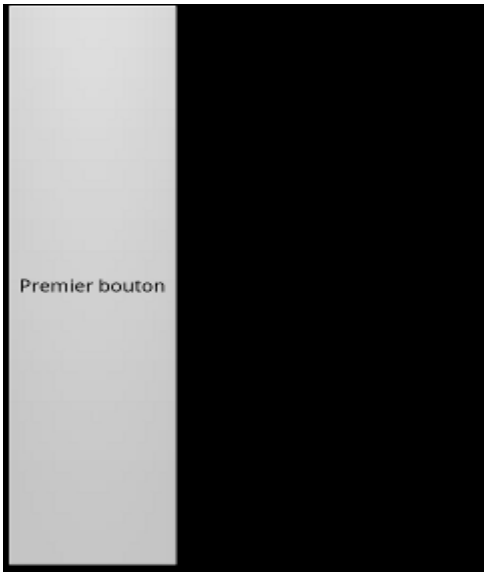
Après la conversation téléphonique, l'utilisateur décide de fermer les activités des deux applications YouTube et Facebook. Ces activités exécutent **onDestroy()**.

Solution 3

- **Le layout 1 :**



- **Le layout 2 :**



Solution 5

N ligne	Origine	Correction
4	Manque de l'appel onCreate () de super Class	Ajouter super.onCreate(savedInstanceState);
5	Manque le nom de la classe R	R.layout.activity_main
9	Paramètre de type View pour onClick ()	onClick(View v)
16	2ème param de constructeur nom de la classe java	Activity2.class
19	Nom de méthode erroné	putExtra (...)
20	Paramètre erroné	`startActivity (i);`

```
public void lancer() {

      EditText e = (EditText) findViewById(R.id.editText1);
      String c = e.getText().toString();
      Intent i = new Intent(Intent.ACTION_VIEW,Uri.parse(c));
      startActivity(i);
 }
```

Solution 7

```
EditText ET1 = (EditText) findViewById(R.id.edit_text_one);
EditText ET2  = (EditText) findViewById(R.id.edit_text_two);
int firstNumber = Integer.parseInt(ET1.getText().toString());
//ou bien int firstNumber = getValue(ET1.getText().toString());
int secondNumber = Integer.parserInt(ET2.getText().toString());
int sum = firstNumber + secondNumber;
TVRes.setText(String.valueOf(sum));   //valueOf () converts int to string
public class MainActivity extends AppCompatActivity {
    Button btn;
    TextView tv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView (R.layout.activity_main);

        btn = (Button) findViewById(R.id.BAdd);
        tv = (TextView )findViewById(R.id.txtResult);
        EditText ET1 = (EditText)
                    findViewById(R.id.edit_text_one);
        EditText ET2  = (EditText)
                    findViewById(R.id.edit_text_two);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick (View v) {
           int firstNumber =
                Integer.parseInt(ET1.getText().toString());
           //ou bien int firstNumber = getValue(ET1.getText().toString());
           int secondNumber =
                      Integer.parserInt(ET2.getText().toString());
           int sum = firstNumber + secondNumber;
           TVRes.setText(String.valueOf(sum));
```

```
            //valueOf () converts int to string
    }
        });
}
```

Solution 8

```
public class MainActivity extends Activity {
    Button btn;
      Intent i = new Intent (this, SecondActivity.class);
       // ou bien  Intent i = new Intent (getApplicationContext(), SecondActivity.class);
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView (R.layout.activity_main);

        btn = (Button) findViewById(R.id.BAdd);
        tv = (TextView )findViewById(R.id.txtResult);
        EditText ET1 = (EditText)
                findViewById(R.id.edit_text_one);
        EditText ET2  = (EditText)
                findViewById(R.id.edit_text_two);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick (View v) {
          int firstNumber =
  Integer.parseInt(ET1.getText().toString());
          int secondNumber =
  Integer.parserInt(ET2.getText().toString());
          int sum = firstNumber + secondNumber;
          i.putExtra ("val", sum);
          startActivity (i);
           }
        });
```

```
}
public class SecondActivity extends Activity {
    Button b;
    TextView tv;
    Intent ii;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView (R.layout.layout2);
        tv = (TextView )findViewById(R.id.txtResult);
        ii = getIntent ();
       Integer x = ii.getIntExtra (''val'');
       Tv. setText (String.valueOf(x));

       }
      public void retour (View v) {
     // sur xml : <Button ... android : onClick = ''retour'' ..../>
     Intent t = new Intent(getApplicationContext(),
                              MainActivity.class);
          startActivity (t);
     }
        }

public class SecondActivity extends Activity {
    Button b;
    TextView tv;
    Intent ii;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView (R.layout.layout2);
        tv = (TextView )findViewById(R.id.txtResult);
        ii = getIntent ();
```

```
        Integer x = ii.getIntExtra (''val'');
        Tv. setText (String.valueOf(x));

        }
       public void retour (View v) {
      // sur xml : <Button ... android : onClick = ''retour'' ..../>
      Intent t = new Intent(getApplicationContext(),
                                MainActivity.class);
            startActivity (t);
      }
       public void afficher (View v) {
      // sur xml : <Button ... android : onClick = 'afficher ..../>
      Uri U = Uri.parse (''https://www.android.com'');
      Intent a = new Intent (Internet.ACTION_VIEW, U);
            startActivity (a);
      }
 }
```

Solution 10

1. public classe DBLocation extends SQLiteOpenHelper {

….

Public void onCreate(SQLiteDatabase db) {

String requete=”CREATE TABLE voiture (id INTEGER PRIMARY KEY AUTOINCREMENT, marque TEXT, type TEXT, nbplace INTEGER, AnneeCirculation INTEGER);”;

db.execSQL(requete);

}

}

2. DBLocation DBLoc = new DBLocation(getbaseContext(), ''LOCVOITURE.db'',null,1) ;

SQLiteDatabase db=DBLoc.getWritableDatabase() ;

3.
```
db.execSQL(''INSERT INTO voiture
     (marque,type,nbplace,anneecirculation) VALUES 'Renault'' ,
                                        ''symbole'', 5, 2016) ;  '') ;
```

4.
```
ContentValues values = new ContentValues () ;

values.put (''Marque'', ''Renault'') ;

values.put (''Type'', symbole) ;

values.put (''nbPlaces'', 5) ;

values.put (''AnneeCirculation'', 2016) ;

long i = db.insert (''voiture'', null, values) ;
```

5.
```
int r = db.delete (''voiture'', ''anneeCirculation = ?'', new
String [ ] {String.valueOf (2013) }) ;
```

6. Le code permet d'afficher toutes les lignes de la table voiture l'une après l'autre.

Les mini-projets

Mini-projet 1
(Application Quiz)

L'objectif état de réaliser une application Quiz qui satisfait les exigences suivantes :
- L'application doit comporter une seule activité.
- L'interface graphique doit présenter les éléments suivants :
    - Le nom de l'application affiché en haut.
    - Deux questions dans deux domaines différents (ex : sport et science) avec plusieurs choix.
    - Le score s'affiche au-dessous
    - Si le score est nul, un WebView apparaît pour inviter le joueur à visiter le site wikipedia pour se renseigner au mieux sur les questions proposées.
    - N'oublier pas de déclarer la permission d'accès à Internet dans le fichier AndroidManifest.xml de l'application.

Modifier l'application Quiz du mini-projet 1 pour supporter le scénario suivant :
- L'application doit contenir trois activités :
    - L'activité1 : affiche le nom de l'application, la question 1 et un bouton suivant pour passer à la deuxième activité.
    - L'activité 2 : affiche la question 2 et un bouton valider pour passer à la dernière activité.
    - L'activité 3 : affiche le score obtenu et au cas où le score est nul, un TextView apparaît dont le texte est ''click here to visite www.wikipedia.org'' et quand l'utilisateur clique dessus, le contenu web de la page wikipedia s'affiche. Utilisez dans ce cas un intent implicite avec l'action : android.intent.action.ACTION_VIEW.

- Modifier l'application Quiz pour intègrer un Broadcast Receiver destiné à la réception de l'Intent système signalant l'évènement faible batterie.
    - L'action à associée au récepteur est : android.intent.action.BATTERY_LOW
    - A l'apparition de l'évènement (quand le niveau de batterie devient faible) et le système android diffuse l'intent qui convient, le récepteur affiche un Toast à l'utilisateur pour lui demander d'arrêter de jouer.
- Il est demandé d'implémenter et d'enregistrer le Broadcast Receiver avec les deux méthodes possibles (xml et java).

- Notre application doit maintenant contenir une base de données pour stocker un ensemble de questions (10 questions) et leurs réponses possibles.

- Une question et ses choix s'affichent.
- Deux boutons sont placés au-dessous : bouton ''next'' pour passer à la question suivante, le bouton ''submit responses'' pour finir le test et afficher le score dans une activité à part.

- Toujours avec notre application Quiz. Cette fois-ci, on veut ajouter un Timer géré par un service pour signaler le timeout pour répondre à toutes les questions du quiz.

- Si le timeout est atteint et l'utilisateur n'a pas encore terminé de répondre à toutes les questions, un Toast s'affiche pour indiquer la fin du temps imparti. Ainsi, un bouton Restart apparaît pour permettre à l'utilisateur de rejouer à nouveau.
- Si par contre l'utilisateur termine avant le timeout, le Timer doit s'arrêter immédiatement (là il faut préciser les manipulations nécessaires pour le service qui gère le Timer).

Mini-Projet 2
(Application mobile pour la gestion des besoins urgents dans les cas exceptionnels (épidémies, catastrophes naturelles..)

La situation épidémiologique mondiale causée par le virus COVID-19 a soulevé de nouveaux problèmes sérieux aussi bien sur le secteur médical que sur les secteurs socioéconomiques.
L'objectif étant de réaliser une application mobile pour la gestion des besoins urgents (besoin de médicaments, prise en charge médicale, aliments...) dans le cas du confinement où les gens se retrouvent isolés les uns des autres en plus de l'interdiction de sortir et/ou se rassembler en dehors de leurs maisons, etc.

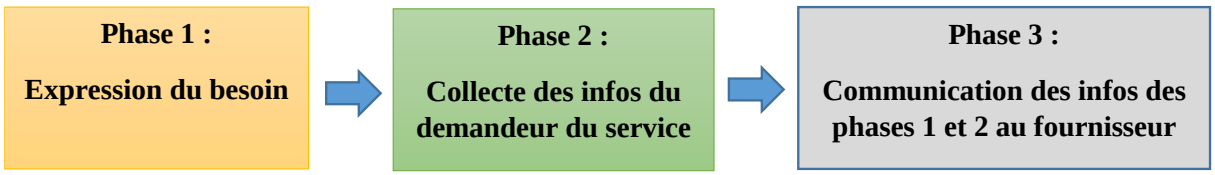
- L'activité principale de l'application doit permettre à l'utilisateur d'exprimer le service demandé.
- La deuxième activité présente un formulaire pour acquérir les informations sur l'identité, numéro de téléphone et adresse pour la livraison du service demandé.
- La validation de l'opération se fait via l'envoi d'un SMS (ou appel téléphonique...), tel que le destinataire peut être soit la gendarmerie, le croissant rouge ou toute autre association).

- Remarque 1
La communication avec le fournisseur de services est supposée être gratuite.

- Remarque 2

Dans le but d'inciter l'esprit créatif chez l'étudiant, nous vous encourageons à spécifier vous-mêmes les détails des différents composants de l'application.

Le schéma (abstrait) suivant résume les étapes de fonctionnement de l'application :



Mini-projet 3
(Application mobile pour gérer le don du sang)

- L'objectif du mini-projet étant de concevoir et réaliser une application mobile Android qui gère le don du sang, dans une ville donnée.

- Pour ce faire, l'application doit pouvoir afficher la liste des groupes sanguins manquants à l'instant (l'information est supposée être fournie par l'hôpital de la ville où se trouve l'utilisateur de l'application).

- L'utilisateur consulte le rapport instantané de l'application et au cas où il accepte (lui-même ou un autre membre de sa famille) et décide de donner du sang, il le signale à travers l'application. Cela se matérialise avec :

- Le remplissage et l'envoi d'un formulaire pour collecter les informations nécessaire sur l'utilisateur (ex : âge (entre 18-70), Poids (> 50kg), autres conditions sur le lien : https://www.maxisciences.com/don-du-sang/don-du-sang-quelles-sont-les-conditions-pour-etre-donneur_art40733.html)
- La communication du formulaire se fait via envoi d'un SMS à un numéro vert dédié à ce service, ou par l'envoi d'un e-mail.

-En réponse à l'acceptation de l'utilisateur, une équipe mobile chargée par l'hôpital se déplace vers l'endroit où habite l'utilisateur, pour effectuer la prise du sang.

Travail demandé :

- Il vous est demandé de concevoir et réaliser, à votre propre style, l'application qui répond à de tel besoin.

Bibliographie

[1] https://developer.android.com/guide. (Le site officiel du développement des Applications Mobiles Android).

[2] L. Gleason, F. Sproviero, V. Gonda, Android Test-Driven Development by Tutorial, Livre édité par: Razeware LLC, 2019.

[3] J Horton, "Android Programming for Beginners: Build in-depth, full-featured Android 9 Pie apps starting from zero programming experience, 2nd Edition", Livre édité par Packt, 2018.

[4] J. F Jerome Demarzio, Android a programmer's guide, Livre édité par : MC Graw Hill, 2016.

Annexe 1 : installation de l'outil de développement : Android Studio

❖ **Conditions liées au système d'exploitation :**

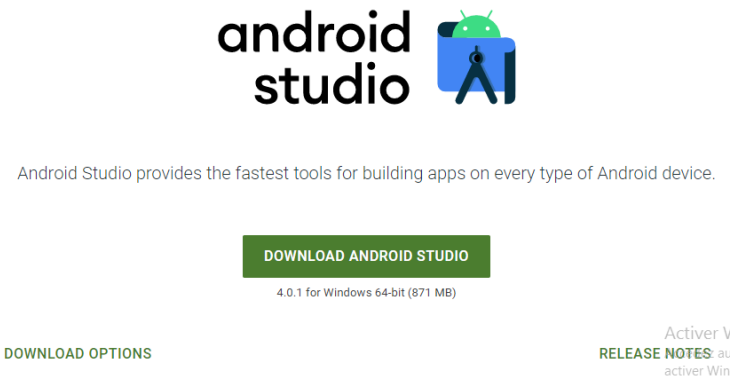
- Microsoft Windows 7/8/10 (64 bits)
- Mac OS à partir de la version 10.8.5
- Une distribution Ubuntu plus récente que la 14.04.

❖ **Conditions matérielles :**

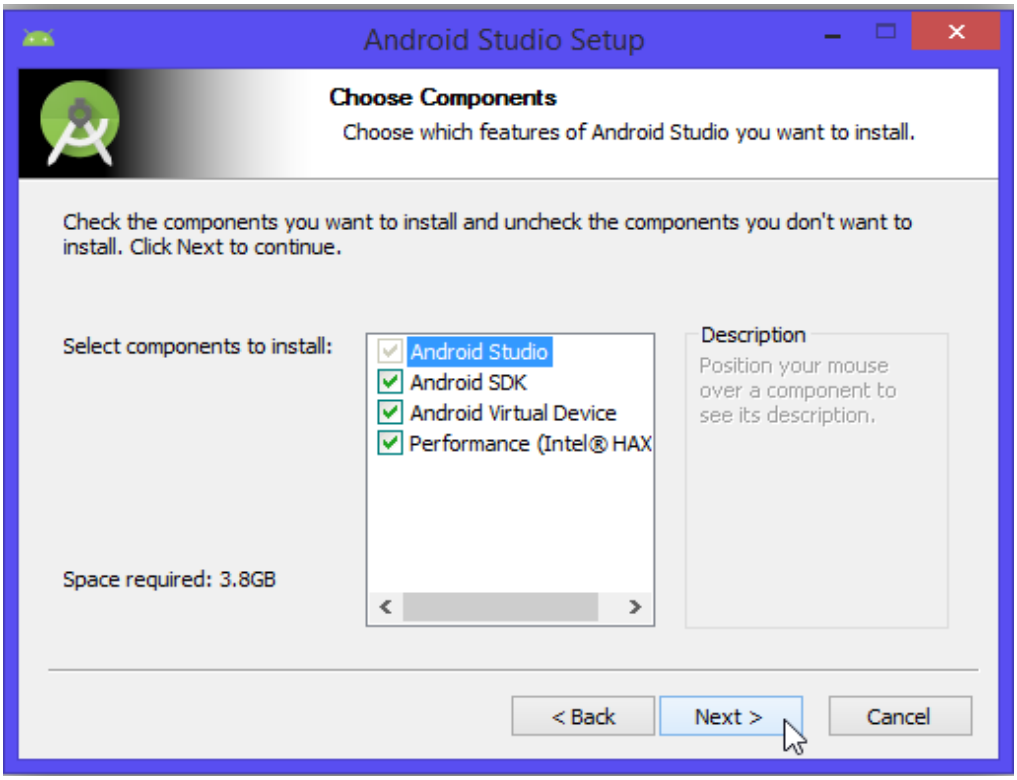
- 4 Go de RAM minimum, 8 Go de RAM recommandés.
- 2 Go d'espace disque disponible au minimum, 4 Go recommandés (500 Mo pour l'IDE + 1,5 Go pour le SDK Android et l'image système de l'émulateur)
- Résolution d'écran minimale de 1280 x 800.

Etape1 : téléchargement de Android Studio

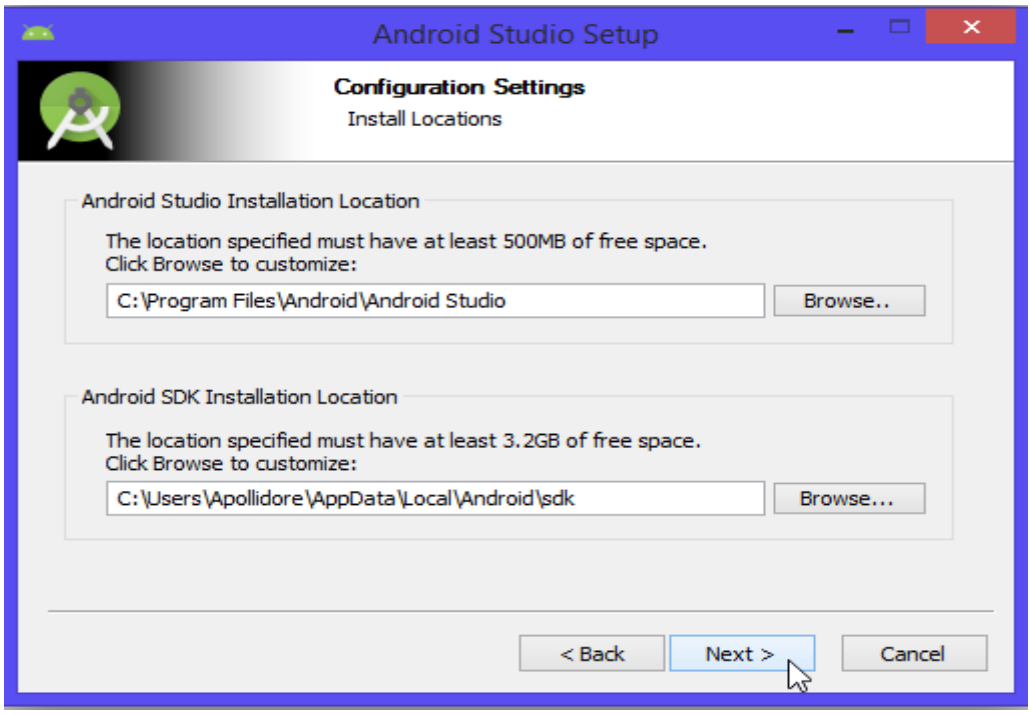
- Avant d'aller installer Android Studio, il faut avoir le JDK (*Java Developement Kit*) bien installé sur la machine.
- Pour installer Androi Studio, il se rendre sur ce site : https://developer.android.com/studio, puis cliquer sur **Download Android Studio**.



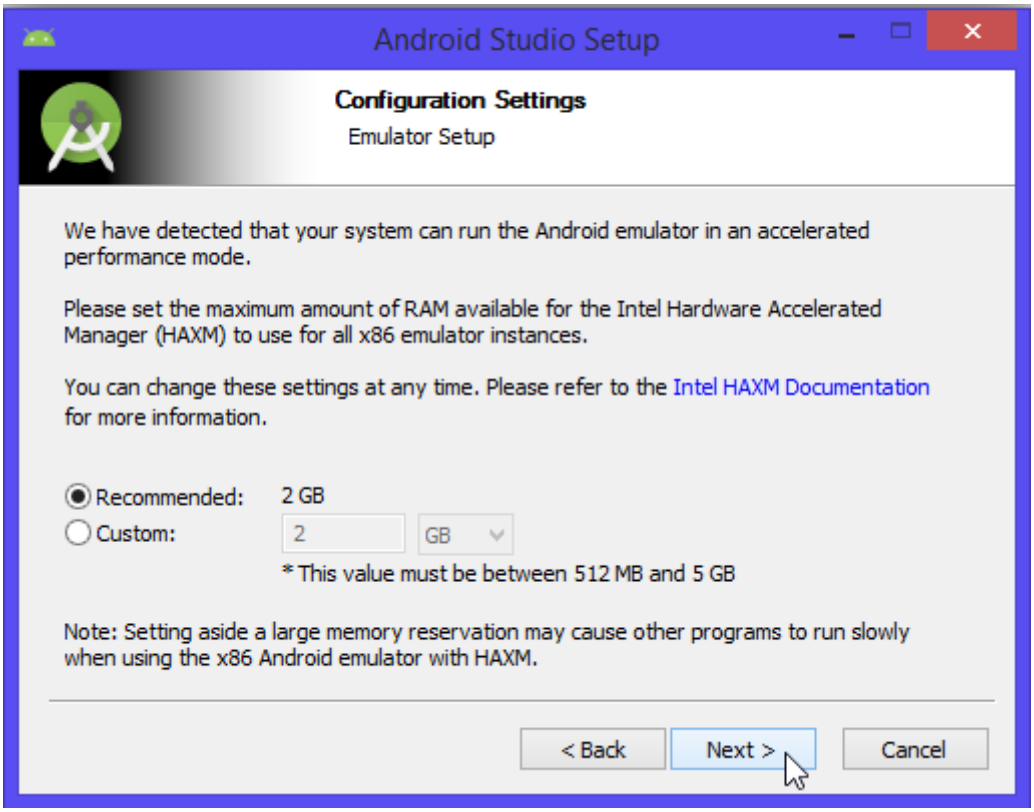
- Une fois que le téléchargement soit terminé, lancez l'installation et conservez les options Android SDK et Android Virtual Device.



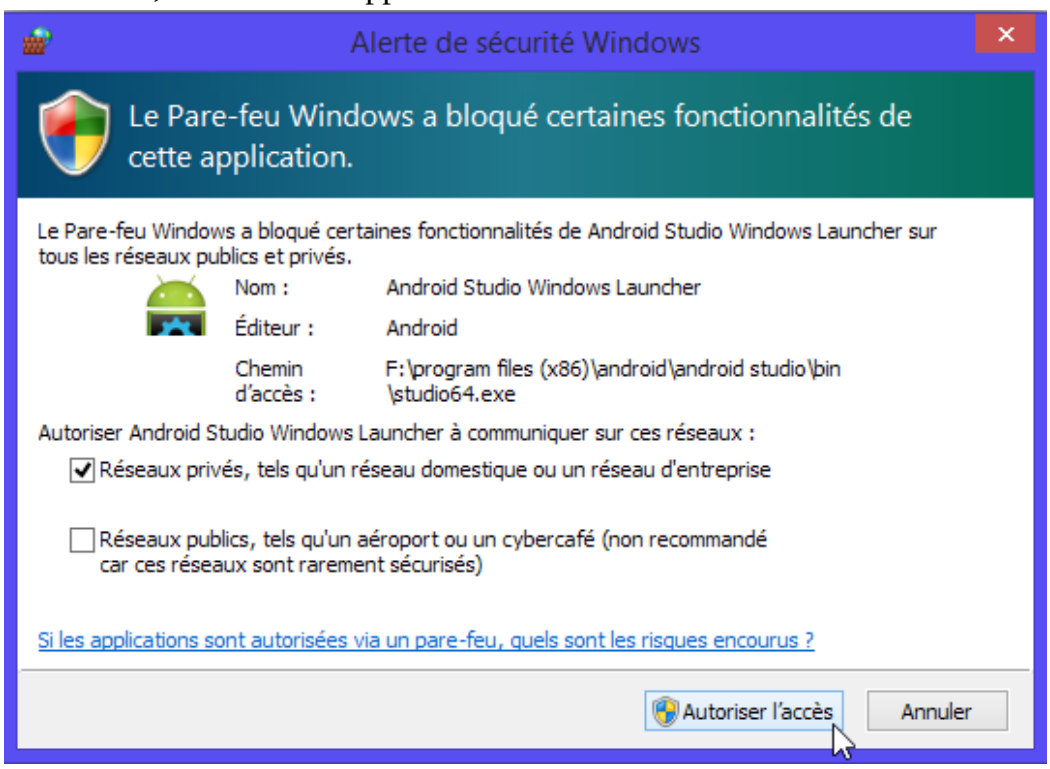
- Puis l'écran ci-dessous s'affiche pour vous permettre d'indiquer où vous souhaitez que Android Studio et le SDK soient installés, puis cliquez sur **Next**.



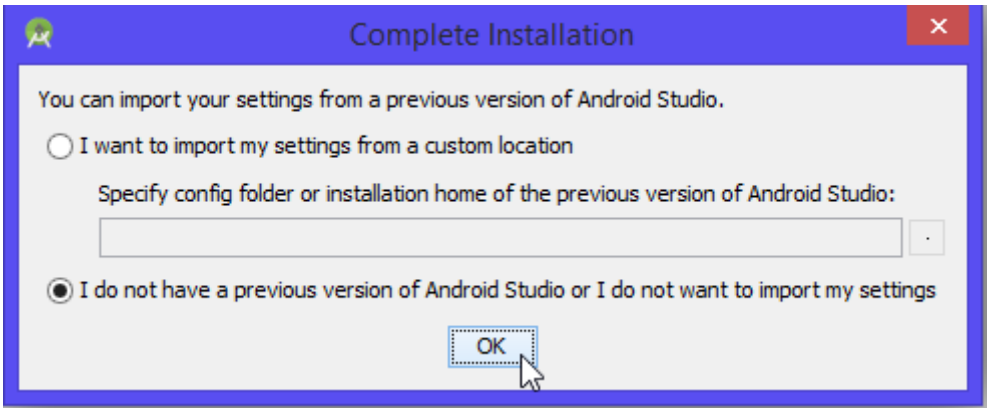
- Si votre ordinateur est suffisamment puissant, l'écran suivant vous propose de montrer quelle quantité de mémoire vive (RAM) vous souhaitez accorder à l'émulateur Android. La valeur par défaut est 2 Go mais vous pouvez indiquer une valeur en choisissant **Custom**. Une fois votre sélection faite, cliquez sur **Next.**



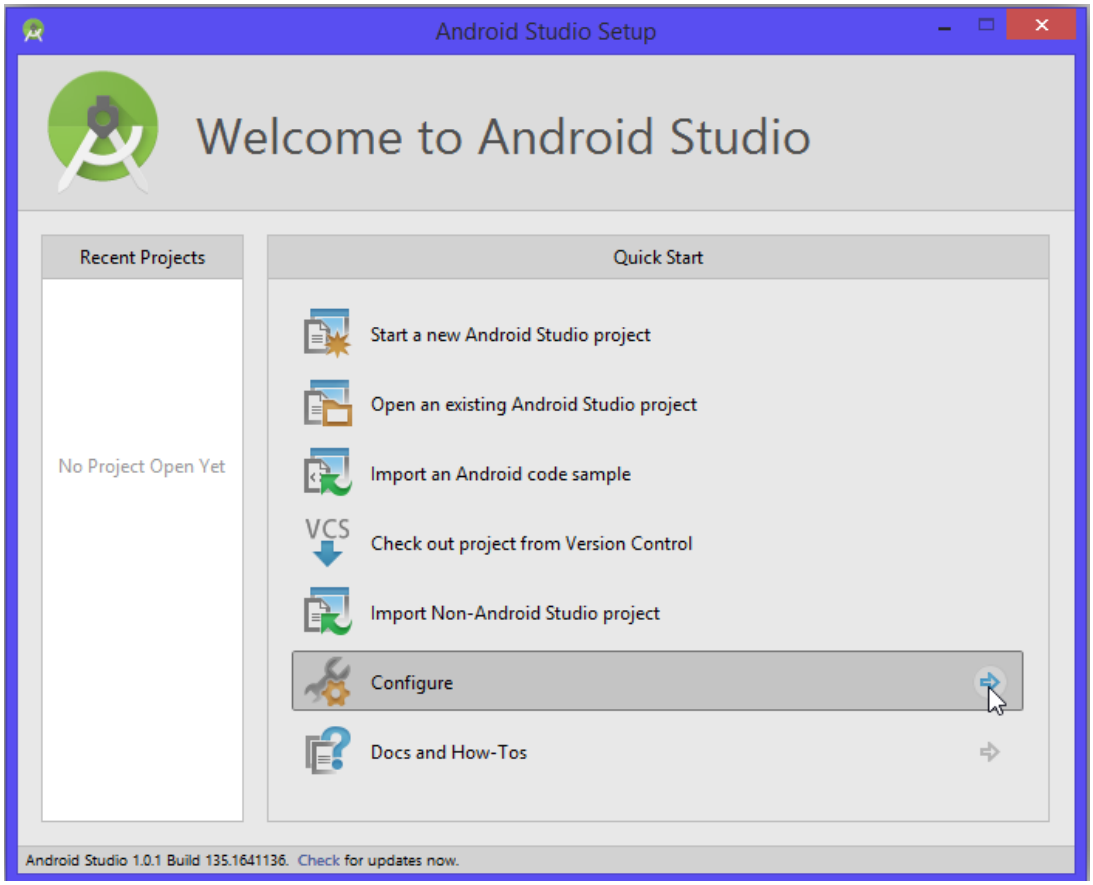
- Une fois l'installation terminée, lancez Android Studio.
- Sous Windows, cette fenêtre apparaît. Il suffit de choisir **Autoriser l'accès**.



- Au premier lancement, une boîte de dialogue s'affiche et vous demande si une version d'Android Studio est déjà installée. Si ce n'est pas le cas, sélectionnez la seconde option :

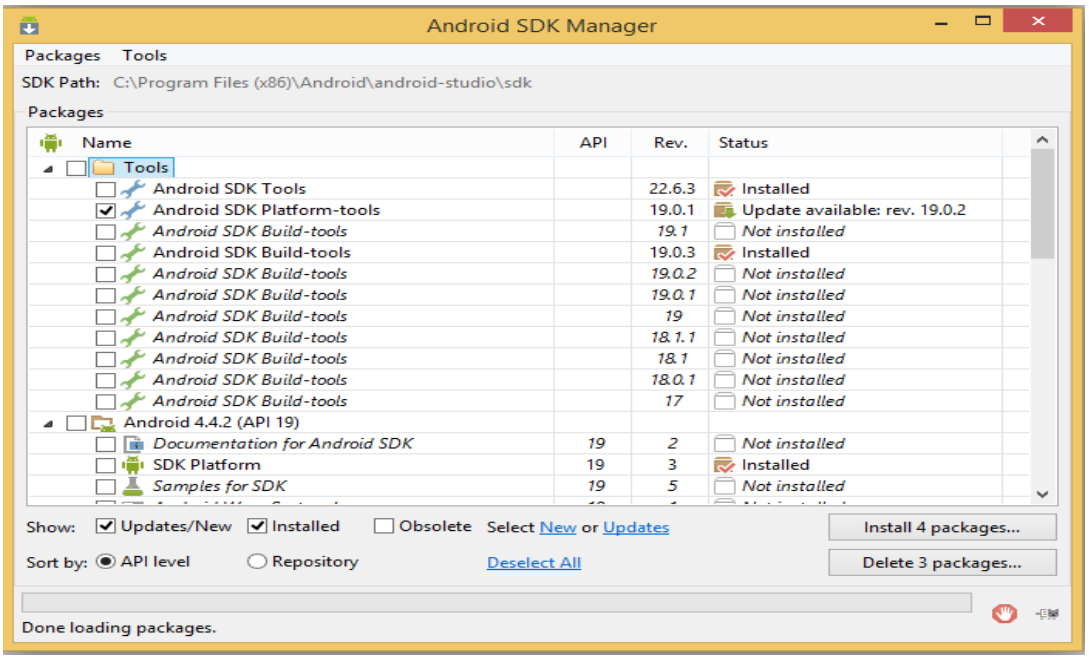


- Une fenêtre s'ouvrira par la suite pour nous demander ce que nous souhaitons faire. On va commencer le télécharger du SDK d'Android. Pour cela, cliquez sur **Configure**.

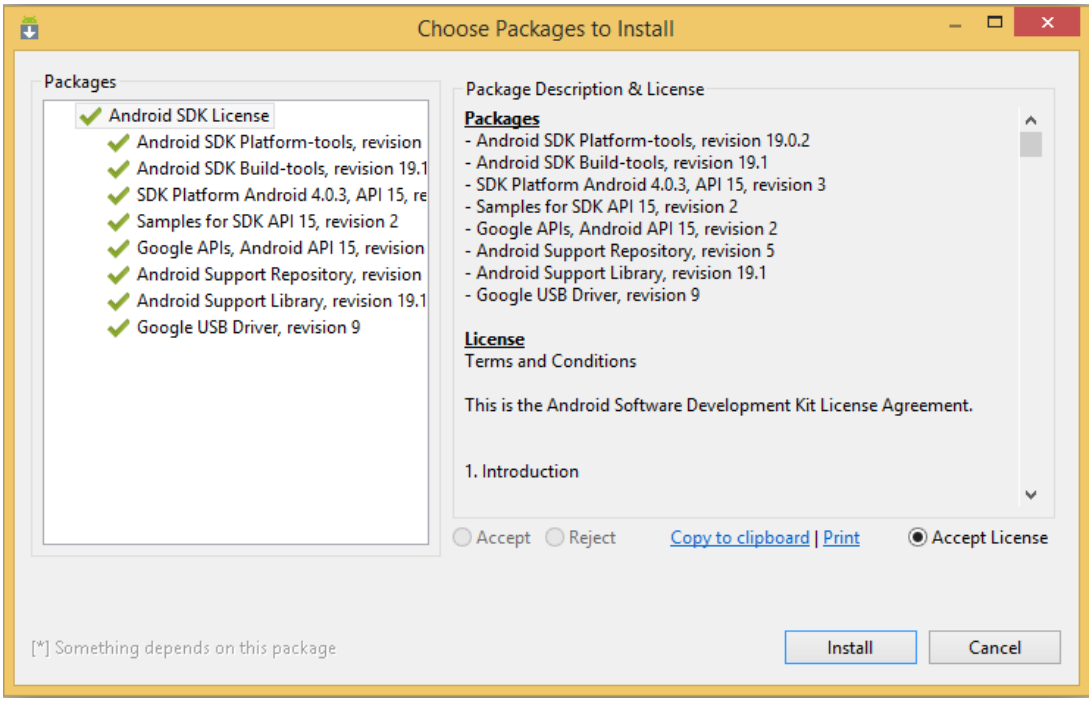


- Cliquez ensuite sur **SDK Manager**.

- Le Android SDK Manager s'ouvre et nous aurons un écran similaire à celui-ci :



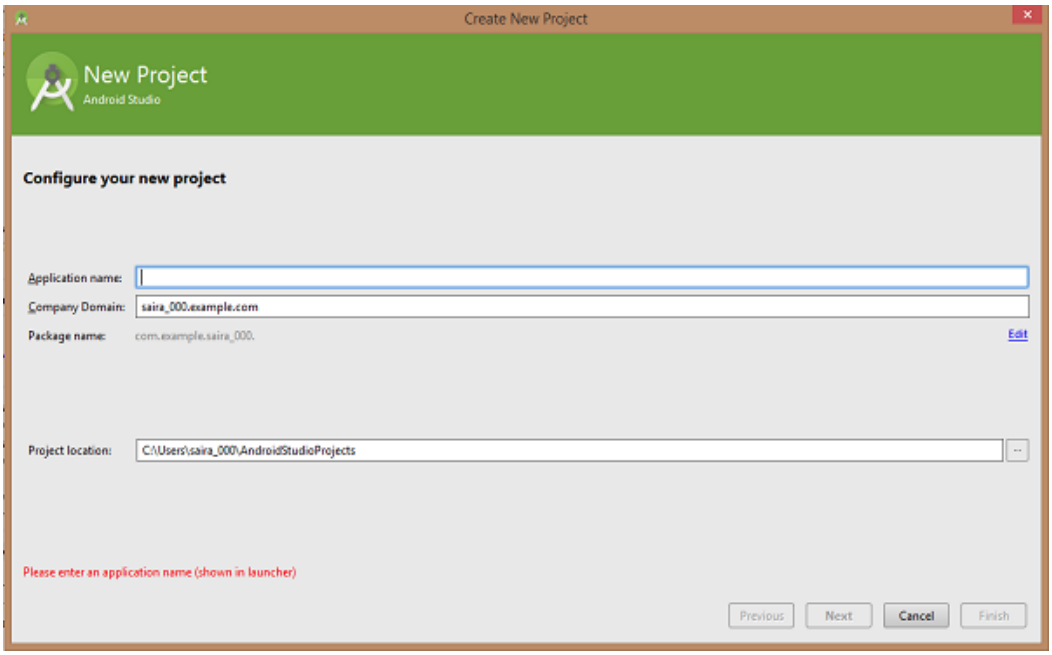
- A ce niveau, il est nécessaire de cocher puis installer un maximum de paquets.
- Finalement, il faut valider les licences pour les fichiers téléchargés


Choose Packages to Install
Packages
Android SDK License
Android SDK Platform-tools, revision
Android SDK Build-tools, revision 19.1
SDK Platform Android 4.0.3, API 15, re
Samples for SDK API 15, revision 2
Google APIs, Android API 15, revision
Android Support Repository, revision
Android Support Library, revision 19.1
Google USB Driver, revision 9
Package Description & License
Packages
- Android SDK Platform-tools, revision 19.0.2
- Android SDK Build-tools, revision 19.1
- SDK Platform Android 4.0.3, API 15, revision 3
- Samples for SDK API 15, revision 2
- Google APIs, Android API 15, revision 2
- Android Support Repository, revision 5
- Android Support Library, revision 19.1
- Google USB Driver, revision 9
License
Terms and Conditions
This is the Android Software Development Kit License Agreement.
1. Introduction
Accept
Reject
Copy to clipboard | Print
Accept License
[*] Something depends on this package
Install
Cancel

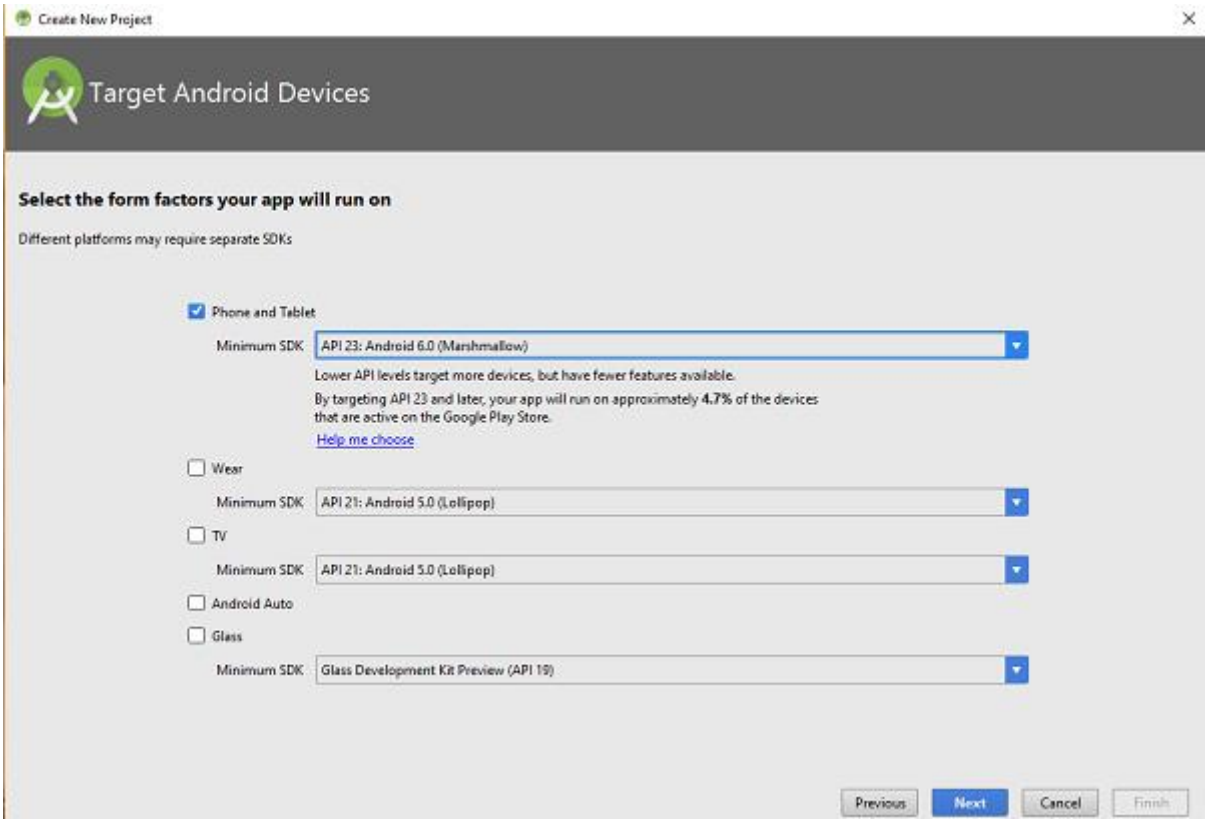
Annexe 2 : La première application mobile Android

La première étape consiste à créer une application Android simple via Android studio. Lorsque vous cliquez sur l'icône du studio Android, l'écran affichera comme montré dans la figure ci-dessous.

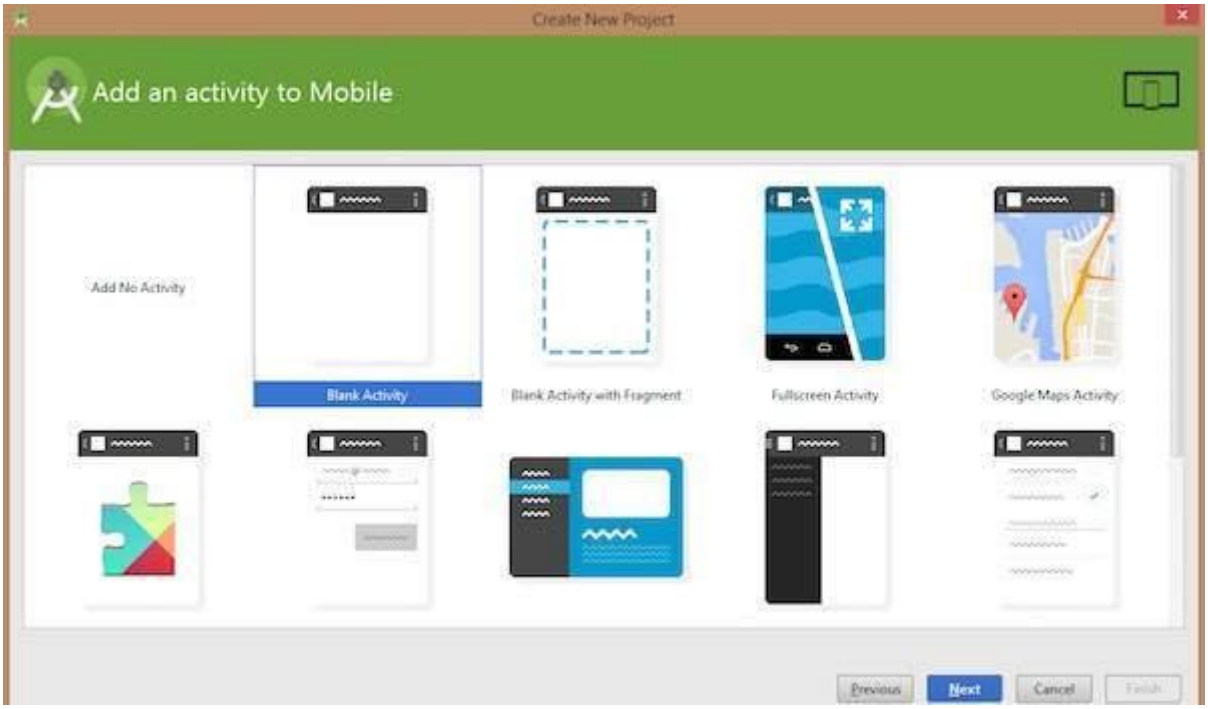
Vous pouvez démarrer le développement de votre application en appelant démarrer un nouveau projet de Android studio. Un nouveau cadre d'installation devrait demander le nom de l'application, les informations sur le package et l'emplacement du projet.



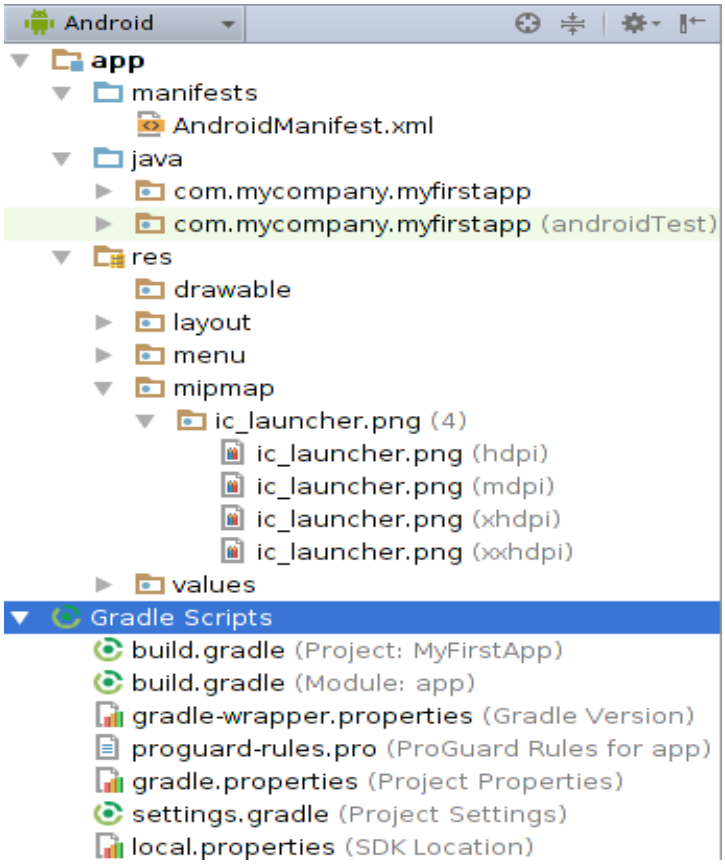
Après avoir entré le nom de l'application, il va être appelé sélectionnez les facteurs de forme sur lesquels votre application fonctionne, ici vous devez spécifier le SDK minimum, dans l'exemple on a déclaré API23.



Le niveau d'installation suivant doit contenir la sélection de l'activité sur mobile, il spécifie la disposition par défaut pour les applications.



Au stade final, il s'agira du développement proprement dit pour écrire le code de l'application en ses différentes parties java et xml.



Après le choix de l'AVD (*Android Virtual device*: dans notre cas c'est généralement un smartphone) lors de la configuration de l'environnement. Et afin d'exécuter l'application à partir de Android studio, il faut ouvrir l'un des fichiers d'activité du projet et cliquer sur l'icône exécuter dans la barre d'outils. Android studio installe l'application sur l'AVD et la démarre. Si tout va bien avec la configuration et l'application, on nous afficherait la fenêtre de l'émulateur similaire à la figure en bas :


5:00
Hello world !

Annexe 3 : Les permissions que peut demander une application mobile Android

Les autorisations normales

Les autorisations normales couvrent les zones où votre application a besoin d'accéder à des données ou à des ressources de l'application, mais où il y a très peu de risque pour la confidentialité de l'utilisateur ou le fonctionnement d'autres applications.

- ACCESS_LOCATION_EXTRA_COMMANDS
- ACCESS_NETWORK_STATE
- ACCESS_NOTIFICATION_POLICY
- ACCESS_WIFI_STATE
- BLUETOOTH
- BLUETOOTH_ADMIN
- BROADCAST_STICKY
- CHANGE_NETWORK_STATE
- CHANGE_WIFI_MULTICAST_STATE
- CHANGE_WIFI_STATE
- DISABLE_KEYGUARD
- EXPAND_STATUS_BAR
- FOREGROUND_SERVICE
- GET_PACKAGE_SIZE
- INSTALL_SHORTCUT
- INTERNET
- KILL_BACKGROUND_PROCESSES
- MANAGE_OWN_CALLS
- MODIFY_AUDIO_SETTINGS
- NFC
- READ_SYNC_SETTINGS
- READ_SYNC_STATS
- RECEIVE_BOOT_COMPLETED
- REORDER_TASKS
- REQUEST_COMPANION_RUN_IN_BACKGROUND
- REQUEST_COMPANION_USE_DATA_IN_BACKGROUND
- REQUEST_DELETE_PACKAGES
- REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
- SET_ALARM
- SET_WALLPAPER
- SET_WALLPAPER_HINTS
- TRANSMIT_IR
- USE_FINGERPRINT
- VIBRATE
- WAKE_LOCK
- WRITE_SYNC_SETTINGS

Les autorisations de signature

Le système accorde ces autorisations d'application au moment de l'installation, mais uniquement lorsque l'application qui tente d'utiliser une autorisation est signée par le même certificat que l'application qui définit l'autorisation (même développeur différentes applications)

- BIND_ACCESSIBILITY_SERVICE
- BIND_AUTOFILL_SERVICE
- BIND_CARRIER_SERVICES
- BIND_CHOOSER_TARGET_SERVICE
- BIND_CONDITION_PROVIDER_SERVICE
- BIND_DEVICE_ADMIN
- BIND_DREAM_SERVICE
- BIND_INCALL_SERVICE
- BIND_INPUT_METHOD
- BIND_MIDI_DEVICE_SERVICE
- BIND_NFC_SERVICE
- BIND_NOTIFICATION_LISTENER_SERVICE
- BIND_PRINT_SERVICE
- BIND_SCREENING_SERVICE
- BIND_TELECOM_CONNECTION_SERVICE
- BIND_TEXT_SERVICE
- BIND_TV_INPUT
- BIND_VISUAL_VOICEMAIL_SERVICE
- BIND_VOICE_INTERACTION
- BIND_VPN_SERVICE
- BIND_VR_LISTENER_SERVICE
- BIND_WALLPAPER
- CLEAR_APP_CACHE
- MANAGE_DOCUMENTS
- READ_VOICEMAIL
- REQUEST_INSTALL_PACKAGES
- SYSTEM_ALERT_WINDOW
- WRITE_SETTINGS
- WRITE_VOICEMAIL

Les autorisations dangereuses

Les autorisations dangereuses couvrent les zones où l'application souhaite des données ou des ressources qui impliquent les informations privées de l'utilisateur, ou pourraient potentiellement affecter les données stockées de l'utilisateur ou le fonctionnement d'autres applications :

- READ_CALENDAR
- WRITE_CALENDAR
- CAMERA
- READ_CONTACTS
- WRITE_CONTACTS
- GET_ACCOUNTS
- ACCESS_FINE_LOCATION
- ACCESS_COARSE_LOCATION
- RECORD_AUDIO
- READ_PHONE_STATE
- READ_PHONE_NUMBERS
- CALL_PHONE
- ANSWER_PHONE_CALLS
- READ_CALL_LOG
- WRITE_CALL_LOG
- ADD_VOICEMAIL
- USE_SIP
- PROCESS_OUTGOING_CALLS
- ANSWER_PHONE_CALLS
- READ_PHONE_NUMBERS
- BODY_SENSORS
- SEND_SMS
- RECEIVE_SMS
- READ_SMS
- RECEIVE_WAP_PUSH
- RECEIVE_MMS
- READ_EXTERNAL_STORAGE
- WRITE_EXTERNAL_STORAGE
- CALL_LOG
- READ_CALL_LOG
- WRITE_CALL_LOG
- PROCESS_OUTGOING_CALLS