

Chapitre1 : Architecture et fonctionnalités de base de la plate-forme Android.

I. Introduction

Les Smartphones sont des appareils extrêmement sophistiqués, qui fournissent des fonctionnalités en plus de celles des téléphones mobiles classiques comme la navigation sur le web, la consultation et l'envoi de courriers électroniques, la messagerie vocale et visuelle, etc. Les Smartphones exécutent tous divers logiciels et applications grâce à des systèmes d'exploitation spécialement conçus pour les mobiles. Les Smartphones peuvent être personnalisés en y installant des applications additionnelles telles que des jeux ou des utilitaires grâce aux magasins d'applications en ligne (stores).

Il existe certaines contraintes pour le développement et la programmation smartphone, qui ne s'appliquent pas au développement habituel. Au moment, la mémoire RAM sur les téléphones est limitée, ce qui implique qu'on peut lancer moins de logiciels à la fois et donc les logiciels doivent faire en sorte de réserver moins de mémoire. Il est aussi important de prendre en compte que nos applications (programs) doivent pouvoir interagir avec un system complet sans l'interrompre. C'est-à-dire il faut respecter une certaine priorité des tâches, par exemple les systèmes permettent de recevoir des messages et des appels pendant l'utilisation d'une autre application. La variation de la taille des écrans, doit être considérée encore lors de la réalisation des applications pour les smartphones.

II. la plate-forme Android

Il existe plusieurs systèmes d'exploitation spécifiques aux Smartphones. Les systèmes d'exploitation les plus utilisés sont **Android**, **iOS** et **Windows Phone**.

II.1. Qu'est-ce qu'Android ?

Android est un système d'exploitation open source pour smartphones, PDA et terminaux mobiles.... Il est développé par la société mondialement connue : Google pour intégrer les applications Google : Gmail, Google Maps, Google Agenda, YouTube et la géolocalisation..... Pour diffuser en masse son système, Google a fédéré autour d'Android une trentaine de sociétés (dont Sag, Motorola, HTC, Sony Ericsson, LG, ...) à l'intérieur de l'Open Handset Alliance (OHA). Le but de l'OHA est de favoriser l'innovation sur les appareils mobiles en fournissant une plate-forme véritablement ouverte, complète et gratuite.

Le SDK Android (Software Developement Kit) fournit les outils et les API (Applications Programming Interface) nécessaires pour développer des applications sur la plateforme Android en utilisant le langage de programmation Java.

La plateforme Android propose notamment :

- un framework permettant la réutilisation et le remplacement de composants,
- une machine virtuelle optimisée pour les appareils mobiles,
- un navigateur intégré basé sur le moteur open source WebKit,
- un moteur graphique optimisé, propulsé par une librairie 2D dédiée et un moteur 3D basé sur les spécifications OpenGL ES 1.0,
- le système de gestion de base de données SQLite pour le stockage de données,
- un support média pour les principaux formats audio, vidéo et images,
- la téléphonie GSM, les communications Bluetooth, 3G et WiFi,
- un accès à la caméra, au GPS, à la boussole et aux accéléromètres,
- un environnement de développement riche : émulateur, outils de débogage, ...



L'Android Market, renommé Google Play Store en mars 2012, permet le téléchargement d'applications gratuites ou payantes. Il est aussi possible de les noter et de les commenter. Fin 2010, il y avait déjà plus de 100 000 applications sur l'Android Market, dont seulement 35,6 % payantes ; en septembre 2011, on dénombrait sur Google Play Store quelques 520 000 applications dont toujours environ 65 % gratuites... La progression du nombre d'applications sur Google Play Store est exponentielle ! Cette progression s'explique par le développement totalement ouvert d'Android, et les applications peuvent d'ailleurs être distribuées autrement que par ce biais. Pour simplifier le développement d'applications, Google a développé une interface web : App Inventor permettant de développer facilement une application qui pourra ensuite être mise à disposition sur le marché.

II.2. Origine

Android, qui se prononce Androïd, doit son nom à la startup du même nom (spécialisée dans le développement d'applications mobiles), rachetée par Google en août 2005. Nom qui vient lui-même d'« androïde » qui désigne un robot construit à l'image d'un être humain...Fondé sur le noyau Linux, le système d'exploitation Android de Google a creusé l'écart avec Apple dans le secteur des smartphones. Samsung reste actuellement le leader incontesté des ventes d'Android, avec 73,3 millions de smartphones vendus en trois mois et 39,1% de parts de marché.

II.3. Les versions

Les différentes versions ont des noms de dessert (qui suivent l'ordre alphabétique, de A à Z) qui sont sculptés et affichés devant le siège social de Google (Mountain View). D'où vient le nom de dessert apposé sur chaque version d'Android ? Au départ, il s'agit d'un petit délire dans l'équipe en charge du projet. L'idée c'est que les desserts, comme les smartphones et tablettes, sont là pour rendre nos vies plus agréables !!!!

- 1.0 : Version connue des développeurs : sortie avant le premier téléphone Android (fin 2007).
- 1.1 : Version incluse dans le premier téléphone, le HTC Dream
- 1.5 : Cupcake (Petit Gâteau - Avril 2009), API level 3
- 1.6 : Donut (Beignet - Septembre 2009), API level 4
- 2.1 : Eclair (Eclair - Janvier 2010), API level 7
- 2.2 : FroYo (Frozen Yogourt / Yaourt glacé - Mai 2010), API level 8
- 2.3 : Gingerbread (Pain d'épice - Décembre 2010), API level 9
- 3.0 : Honeycomb (Gâteau de Miel – Février 2011), API level 11
 - 3.1 : Mars 2011, API level 12
 - 3.2 : Juillet 2011, API level 13
- 4.0 : Ice Cream Sandwich (Sandwich à la crème glacée - Octobre 2011), API level 14
 - 4.0.3 : Décembre 2011, API level 15
- 4.1 : Jelly Bean (Bonbon à la gelée / Dragée – Juillet 2012), API level 16
 - 4.2 : Octobre 2012, API level 17
 - 4.3 : Juillet 2013, API level 18
- 4.4 : Kitkat, octobre 2013.
- 5.0 : Lollipop, novembre 2014.
- 6.0 : Marshmallow en fin 2015, Nougat, Oreo Puis ANDROID 9 Pie, ANDROID10, ANDROID11.

III. Architecture Android

Architecture en "pile logicielle" :

- **La couche "Applications"** : Android est utilisé dans un ensemble contenant déjà des applications natives comme, un client de mail, des programmes pour envoyer des SMS, d'agenda, de navigateur web, de contacts personnels.....
- **La couche "Application Framework"** : cette couche permet au programmeur de construire de nouvelles applications. Cette couche fournit la gestion :
 - des Views (= IHM).
 - des ContentProviders = l'accessibilité aux données des autres applications (ex: les contacts) et donc les partages de données.
 - des ressources = les fichiers non codes comme les images, les écrans (Resource Manager).
 - des Notifications (affichage d'alerte dans la barre de titre).
 - des Activitys = l'enchaînement des écrans.
- **La couche "Libraries" (bibliothèques)** : couche logicielle basse pour utiliser :
 - les formats multimédia : images, audio et vidéo.
 - les dessins 2D et 3D, bitmap et vectoriel.
 - une base de données SQL (SQLite).

- **L'environnement d'exécution (Android Runtime)** : toute application est exécutée dans son propre processus, dans sa propre Dalvik Virtual Machine. DVM est la machine virtuelle Java pour les applications Android, conçu pour exécuter du code Java pour des systèmes ayant des contraintes de place mémoire et rapidité d'exécution. Elle exécute du code .dex (Dalvik executable) qui sont des .class adaptés à l'environnement Android. Ecrit par Dan Bornstein d'où le nom est un village islandais dont sont originaires certains de ses ancêtres. Elle a été choisie par Google car plusieurs instances de la DVM peuvent être lancées efficacement. Le code de la DVM est open source.
- **Le noyau Linux** : sur lequel la Dalvik virtual machine s'appuie pour gérer le multithreading, la mémoire. Le noyau Linux apporte les services de sécurité, la gestion des processus, etc.

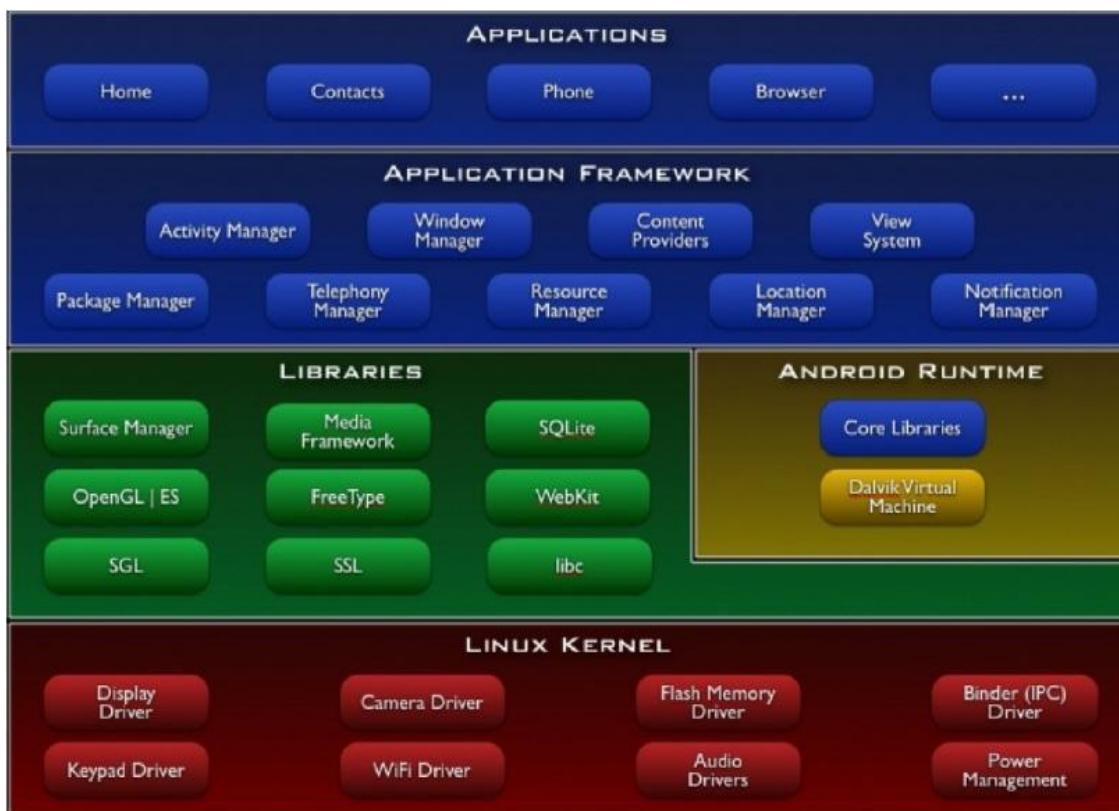


Figure1. Architecture en "pile logicielle".

Chapitre2 : Développement Android.

I. Les composants d'une application Android

Les composants fondamentaux d'une application Android sont : Activity, Service, Content Provider, Broadcast Receiver. Certains de ces composants communiquent entre eux à l'aide d'Intent :

- **Activity** (activité) "gère l'affichage et l'interaction d'un écran" : La plupart des applications se compose de plusieurs écrans. Chaque écran peut être réalisé par une **activité**. Si un nouvel écran s'ouvre, le système utilise une pile d'histoire pour stocker les écrans précédents et pouvoir reprendre l'état précédent ou enlever cet état.
- **Service** : Un Service est utilisé pour réaliser l'application en arrière-plan. C'est-à-dire, cette application peut marcher quand une autre application est en train de s'exécuter comme les services de lecture de musique.
- **Content Provider** (fournisseur de contenu) : Gère des données partageables. C'est le seul moyen d'accéder à des données partagées entre applications. Exemple de fournisseur de contenu : les informations de contact de l'utilisateur du smartphone.
- **Broadcast Receiver** (récepteur d'informations) : est un composant à l'écoute d'informations qui lui sont destinées. Un tel récepteur indique le type d'informations qui l'intéressent et pour lesquelles il se mettra en écoute. Exemple : appel téléphonique entrant, réseau Wi-Fi connecté, informations diffusées par des applications.

Un récepteur n'est pas une IHM mais peut en lancer une (éventuellement petite : une barre de notification), ou peut lancer un service traitant l'arrivée de l'information.

- Un événement (**intent**) est une "intention" à faire quelque chose contenant des informations destinées à un autre composant Android. C'est un message asynchrone. Les activités, services et récepteurs d'informations utilisent les Intents pour communiquer entre eux.

II. Cycle de vie d'une activité (Activity Life cycle) :

Pour développer une application sur Android, on doit comprendre le cycle de vie d'une activité. Le cycle de vie d'une activité est exprimé par la figure suivante :

- L'état **Active**/courant (Running) : L'activité marche en avant-plan.
- L'état **Paused** (en pause) : Cette activité est visible mais elle n'est pas active.
- L'état **Stopped**: Cette activité n'est pas visible. Si une activité est complètement masquée par une autre activité, elle est arrêtée et conserve tous les états. Cependant elle n'est plus visible pour l'utilisateur, sa fenêtre est cachée et elle sera souvent tuée par le système lorsque la mémoire est nécessaire ailleurs.
- L'état **Dead** : Cette activité est terminée ou elle n'a jamais été démarrée. Si une activité est en pause ou arrêtée, le système peut supprimer l'activité de la mémoire, soit par lui demandant de se terminer, ou tout simplement tuer le processus. Quand il est affiché de nouveau à l'utilisateur, il doit être redémarré et restauré à son état antérieur.

➤ Il existe trois boucles principales :

- La durée de vie d'une activité se passe entre le premier appel à `OnCreate ()` et l'appel à `onDestroy ()`. Une activité met en place tous les états globaux dans la méthode `onCreate ()` et libère toutes les ressources restantes à `onDestroy ()`.
- La durée de vie visible d'une activité se passe entre un appel à `onStart ()` et un appel correspondant à `onStop ()`. Dans ce temps, l'utilisateur peut voir l'activité sur l'écran, même si elle n'est pas à l'avant et à l'interaction avec l'utilisateur. Entre ces deux méthodes, les ressources qui sont nécessaires pour montrer l'activité de l'utilisateur sont conservées.
- La durée de vie d'une activité en avant-plan se passe entre un appel à `onResume ()` et un appel correspondant à `onPause ()`. Dans ce temps, l'activité est en face de toutes les autres activités afin d'interagir avec l'utilisateur. Une activité peut souvent changer son état entre l'état de reprise et l'état en pause.

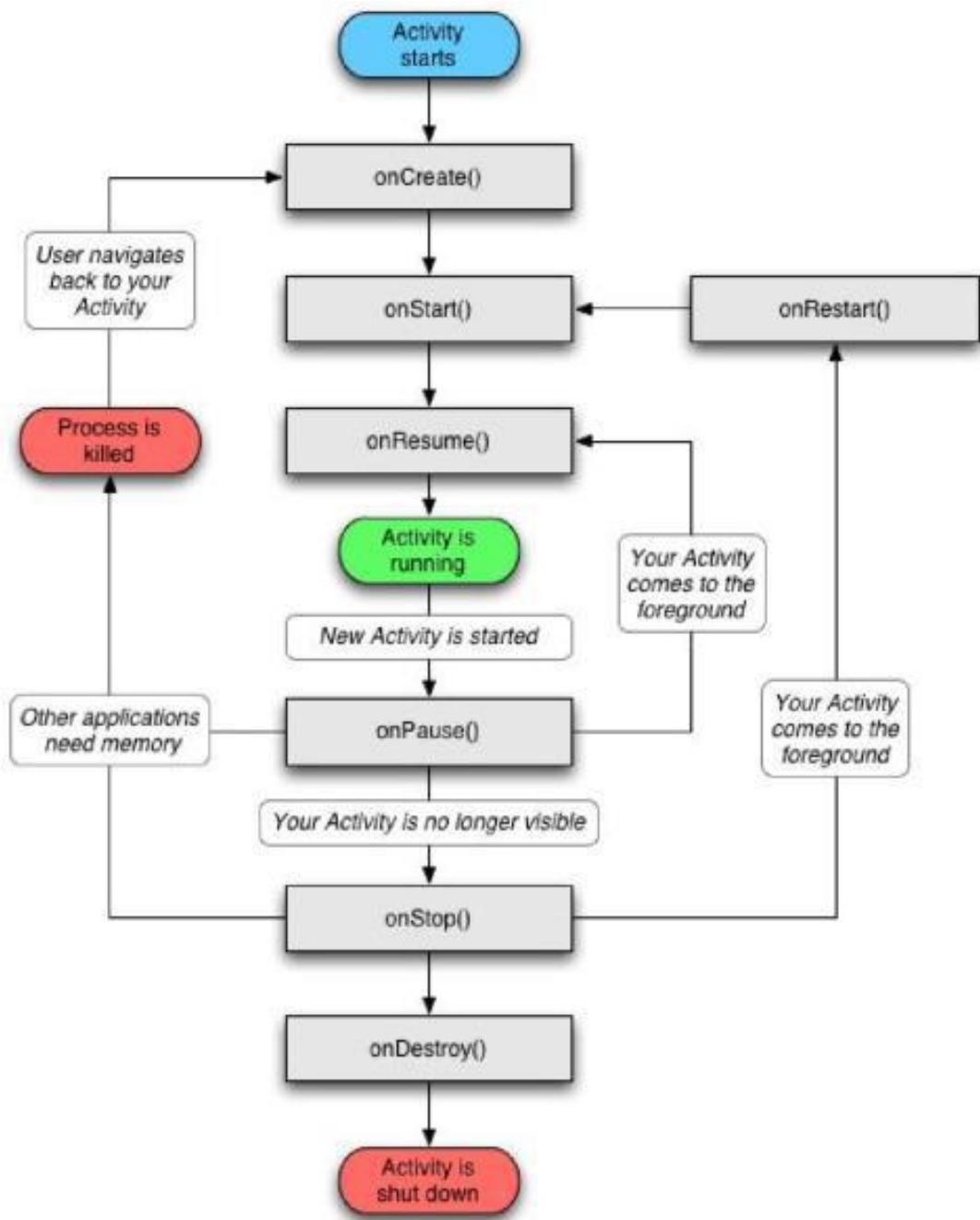


Figure1. Cycle de vie d'une activité.

3

Création d'applications et découverte des activités

Une application Android est un assemblage de composants liés grâce à un fichier de configuration.

Avant de rentrer dans le détail d'une application Android et de son projet associé, différents concepts fondamentaux sont à préciser :

- les activités ;
- les vues et contrôles (et leur mise en page) ;
- les ressources ;
- le fichier de configuration appelé également manifeste.

Les *vues* sont les éléments de l'interface graphique que l'utilisateur voit et sur lesquels il pourra agir. Les vues contiennent des composants, organisés selon diverses mises en page (les uns à la suite des autres, en grille...).

Les *contrôles* (boutons, champs de saisie, case à cocher, etc.) sont eux-mêmes un sous-ensembles des vues (nous y reviendrons ultérieurement). Ils ont besoin d'accéder aux textes et aux images qu'ils affichent (par exemple un bouton représentant un téléphone aura besoin de l'image du téléphone correspondante). Ces textes et ces images seront puisés dans les fichiers *ressources* de l'application.

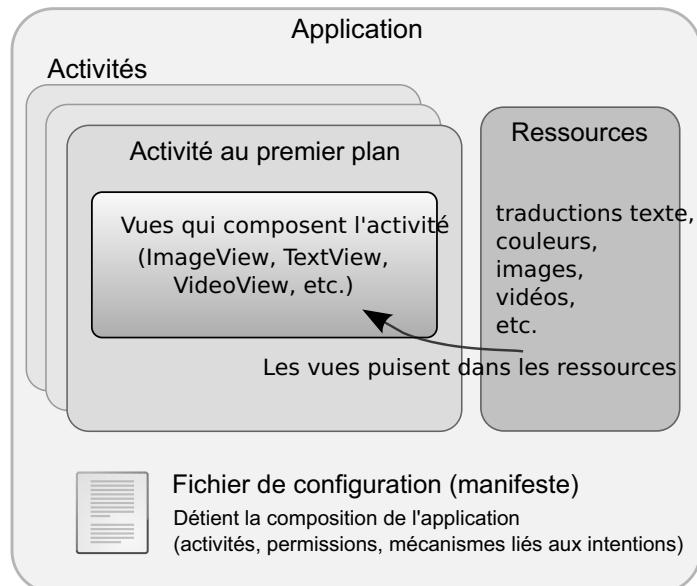
Une *activité* peut être assimilée à un écran structuré par un ensemble de vues et de contrôles composant son interface de façon logique : elle est composée d'une hiérarchie de vues contenant elles-mêmes d'autres vues. Une activité est par exemple un formulaire d'ajout de contacts ou encore un plan Google Maps sur lequel vous ajouterez de l'information. Une application comportant plusieurs écrans, possédera donc autant d'activités.

À côté de ces éléments, se trouve un fichier XML : le *fichier de configuration* de l'application. C'est un fichier indispensable à chaque application qui décrit entre autres :

- le point d'entrée de votre application (quel code doit être exécuté au démarrage de l'application) ;
- quels composants constituent ce programme ;
- les permissions nécessaires à l'exécution du programme (accès à Internet, accès à l'appareil photo...).

Figure 1

Composition d'une application



Le fichier de configuration Android :

la recette de votre application

Chaque application Android nécessite un fichier de configuration : `AndroidManifest.xml`. Ce fichier est placé dans le répertoire de base du projet, à sa racine. Il décrit le contexte de l'application, les activités, les services, les récepteurs d'Intents (*Broadcast receivers*), les fournisseurs de contenu et les permissions.

Structure du fichier de configuration

Un fichier de configuration est composé d'une racine (le tag `manifest` ①) et d'une suite de nœuds enfants qui définissent l'application.

Code 1 : Structure vide d'un fichier de configuration d'une application

```
<manifest ①
    xmlns:android=http://schemas.android.com/apk/res/android ②
    package="fr.domaine.application"> ③
</manifest>
```

La racine XML de la configuration est déclarée avec un espace de nom Android (`xmlns:android` ②) qui sera utile plus loin dans le fichier ainsi qu'un paquetage ③ dont la valeur est celle du paquetage du projet.

Voici le rendu possible d'un manifeste qui donne une bonne idée de sa structure. Ce fichier est au format XML. Il doit donc toujours être :

- bien formé : c'est-à-dire respecter les règles d'édition d'un fichier XML en termes de nom des balises, de balises ouvrante et fermante, de non-imbrication des balises, etc. ;
- valide : il doit utiliser les éléments prévus par le système avec les valeurs prédéfinies.

Profitons-en pour étudier les éléments les plus importants de ce fichier de configuration.

Code 2 : Structure du fichier AndroidManifest.xml extraite de la documentation

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

<manifest>

    <uses-permission /> ①
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />

    <application> ②

        <activity> ③
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service> ④
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver> ⑤
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
```

```
<provider> ⑥
<grant-uri-permission />
<path-permission />
<meta-data />
</provider>

<uses-library />

</application>

</manifest>

<uses-permission> ①
```

Les permissions qui seront déclarées ici seront un prérequis pour l'application. À l'installation, l'utilisateur se verra demander l'autorisation d'utiliser l'ensemble des fonctions liées à ces permissions comme la connexion réseau, la localisation de l'appareil, les droits d'écriture sur la carte mémoire...

<application> ②

Un manifeste contient un seul et unique nœud application qui en revanche contient des nœuds concernant la définition d'activités, de services...

<activity> ③

Déclare une activité présentée à l'utilisateur. Comme pour la plupart des déclarations que nous allons étudier, si vous oubliez ces lignes de configuration, vos éléments ne pourront pas être utilisés.

<service> ④

Déclare un composant de l'application en tant que service. Ici pas question d'interface graphique, tout se déroulera en tâche de fond de votre application.

<receiver> ⑤

Déclare un récepteur d'objets Intent. Cet élément permet à l'application de recevoir ces objets alors qu'ils sont diffusés par d'autres applications ou par le système.

<provider> ⑥

Déclare un fournisseur de contenu qui permettra d'accéder aux données gérées par l'application.

Qu'est-ce qu'une activité Android ?

Une activité peut être assimilée à un écran qu'une application propose à son utilisateur. Pour chaque écran de votre application, vous devrez donc créer une activité. La transition entre deux écrans correspond (comme vu un peu plus haut dans le cycle de vie d'une application) au lancement d'une activité ou au retour sur une activité placée en arrière-plan.

Une activité est composée de deux volets :

- la logique de l'activité et la gestion du cycle de vie de l'activité qui sont implémentés en Java dans une classe héritant de `Activity` (nous reviendrons plus tard sur tous ces concepts) ;
- l'interface utilisateur, qui pourra être définie soit dans le code de l'activité soit de façon plus générale dans un fichier XML placé dans les ressources de l'application.
-

Voici venu le moment de voir à quoi ressemble l'activité la plus simple possible.

Code 3 : Squelette minimal pour créer une première activité

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

public class ActiviteSimple extends Activity {
    /**
     * Méthode appelée à la création de l'activité
     * @param savedInstanceState permet de restaurer l'état
     * de l'interface utilisateur
     */
```

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
}
}

```

Cycle de vie d'une activité

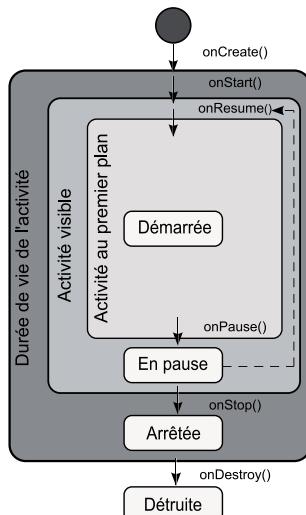
Tout ce que nous avons vu en parlant du cycle de vie d'une application, notamment sur la gestion des processus en fonction des ressources, a un impact direct sur les activités et notamment sur leur cycle de vie.

Les états principaux d'une activité sont les suivants :

- **active (active)** : activité visible qui détient le focus utilisateur et attend les entrées utilisateur. C'est l'appel à la méthode `onResume`, à la création ou à la reprise après pause qui permet à l'activité d'être dans cet état. Elle est ensuite mise en pause quand une autre activité devient active grâce à la méthode `onPause` ;
- **suspendue (paused)** : activité au moins en partie visible à l'écran mais qui ne détient pas le focus. La méthode `onPause` est invoquée pour entrer dans cet état et les méthodes `onResume` ou `onStop` permettent d'en sortir ;
- **arrêtée (stopped)** : activité non visible. C'est la méthode `onStop` qui conduit à cet état.

Voici un diagramme (voir figure 2) qui représente ces principaux états et les transitions y menant.

Figure 2
Cycle de vie d'une activité



Le cycle de vie d'une activité est parsemé d'appels aux méthodes relatives à chaque étape de sa vie. Il informe ainsi le développeur sur la suite des événements et le travail qu'il doit accomplir. Voyons de quoi il retourne en observant chacune de ces méthodes.

Code 4: Squelette d'une activité

```
package com.eyrolles.android.activity;

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

public final class TemplateActivity extends Activity {

    /**
     * Appelée lorsque l'activité est créée.
     * Permet de restaurer l'état de l'interface
     * utilisateur grâce au paramètre savedInstanceState.
     */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Placez votre code ici
    }

    /**
     * Appelée lorsque que l'activité a fini son cycle de vie.
     * C'est ici que nous placerons notre code de libération de
     * mémoire, fermeture de fichiers et autres opérations
     * de "nettoyage".
     */
    @Override
    public void onDestroy(){
        // Placez votre code ici
        super.onDestroy();
    }
    /**
     * Appelée lorsque l'activité démarre.
     * Permet d'initialiser les contrôles.
     */
    @Override
    public void onStart(){
        super.onStart();
        // Placez votre code ici
    }

    /**
     * Appelée lorsque l'activité passe en arrière plan.
     * Libérez les écouteurs, arrêtez les threads, votre activité
     * peut disparaître de la mémoire.
     */
}
```

```
    @Override
    public void onStop(){
        // Placez votre code ici
        super.onStop();
    }

    /**
     * Appelée lorsque l'activité sort de son état de veille.
     */
    @Override
    public void onRestart(){
        super.onRestart();
        //Placez votre code ici
    }

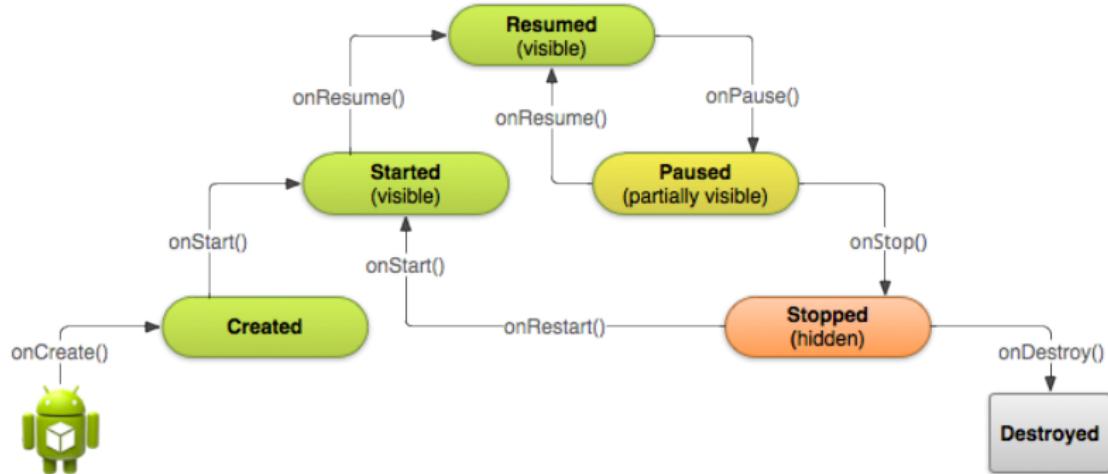
    /**
     * Appelée lorsque que l'activité est suspendue.
     * Stoppez les actions qui consomment des ressources.
     * L'activité va passer en arrière-plan.
     */
    @Override
    public void onPause(){
        //Placez votre code ici
        super.onPause();
    }

    /**
     * Appelée après le démarrage ou une pause.
     * Relancez les opérations arrêtées (threads).
     * Mettez à jour votre application et vérifiez vos écouteurs.
     */
    @Override
    public void onResume(){
        super.onResume();
        // Placez votre code ici
    }

    /**
     * Appelée lorsque l’ activité termine son cycle visible.
     * Sauvez les données importantes.
     */
    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        // Placez votre code ici
        // sans quoi l'activité aura perdu son état
        // lors de son réveil
        super.onSaveInstanceState(savedInstanceState);
    }
```

```
 /**
 * Appelée après onCreate.
 * Les données sont rechargées et l'interface utilisateur.
 * est restaurée dans le bon état.
 */
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    //Placez votre code ici
}
```

Cycle de vie “pyramidal”



Pourquoi implémenter ces méthodes ?

Cela est important afin que votre application fonctionne correctement :

- ▶ Réception d'une appel et basculement sur une autre application ;
- ▶ Ne pas consommer trop de ressources système ;
- ▶ Ne pas avoir de problème lors de la création/restauration de l'application par le système (lors d'une rotation de l'écran par exemple).

Remarque : avant de distribuer une application, Il est donc important d'avoir fait des tests dans les situations évoquées ci-dessus.

Les vues

Les vues sont les briques de construction de l'interface graphique d'une activité Android. Les objets [View](#) représentent des éléments à l'écran qui permettent d'interagir avec l'utilisateur via un mécanisme d'événements.

Plus concrètement, chaque écran Android contient un arbre d'éléments de type [View](#) dont chaque élément est différent de par ses propriétés de forme, de taille...

Bien que la plupart des éléments dont nous ayons besoin – textes, boutons... – soient fournis par la plate-forme, il est tout à fait possible de créer des éléments personnalisés.

Les vues peuvent être disposées dans une activité (objet [Activity](#)) et donc à l'écran soit par une description XML, soit par un morceau de code Java.

Tous ces aspects de création d'interfaces graphiques sont abordés en détails dans le chapitre suivant.

Les ressources

Le but est ici de présenter les différents types de ressources prises en charge et les concepts généraux concernant leur utilisation (syntaxe, format, appel...) dans les projets. La mise en pratique plus concrète de ces notions se fera tout au long des exemples développés sur chaque thème et les concepts plus avancés comme l'internationalisation sont développés plus en détails dans le chapitre traitant des interfaces utilisateur avancées.

À RETENIR Les ressources

Les ressources sont des fichiers externes – ne contenant pas d'instruction – qui sont utilisés par le code et liés à votre application au moment de sa construction. Android offre un support d'un grand nombre de fichiers ressources comme les fichiers images JPEG et PNG, les fichiers XML...

L'externalisation des ressources en permet une meilleure gestion ainsi qu'une maintenance plus aisée. Android étend ce concept à l'externalisation de la mise en page des interfaces graphiques (*layouts*) en passant par celle des chaînes de caractères, des images et bien d'autres...

Physiquement, les ressources de l'application sont créées ou déposées dans le répertoire `res` de votre projet. Ce répertoire sert de racine et contient lui-même une arborescence de dossiers correspondant à différents types de ressources.

Tableau 1 Les types majeurs de ressources avec leur répertoire associé

Type de ressource	Répertoire associé	Description
Valeurs simples	<code>res/values</code>	Fichiers XML convertis en différents types de ressources. Ce répertoire contient des fichiers dont le nom reflète le type de ressources contenues : 1. <code>arrays.xml</code> définit des tableaux ; 2. <code>string.xml</code> définit des chaînes de caractères ; 3. ...
Drawables	<code>res/drawable</code>	Fichiers <code>.png</code> , <code>.jpeg</code> qui sont convertis en bitmap ou <code>.9.png</code> qui sont convertis en "9-patches" c'est-à-dire en images ajustables. Note : à la construction, ces images peuvent être optimisées automatiquement. Si vous projetez de lire une image bit à bit pour réaliser des opérations dessus, placez-la plutôt dans les ressources brutes.
Layouts	<code>res/layout</code>	Fichiers XML convertis en mises en page d'écrans (ou de parties d'écrans), que l'on appelle aussi gabarits.
Animations	<code>res/anim</code>	Fichiers XML convertis en objets animation.
Ressources XML	<code>res/xml</code>	Fichiers XML qui peuvent être lus et convertis à l'exécution par la méthode <code>resources.getXML</code> .
Ressources brutes	<code>res/raw</code>	Fichiers à ajouter directement à l'application compressée créée. Ils ne seront pas convertis.

Toutes ces ressources sont placées, converties ou non, dans un fichier de type `APK` qui constituera le programme distribuable de votre application. De plus, Android crée une classe nommée `R` qui sera utilisée pour se référer aux ressources dans le code.

Création d'interfaces utilisateur 4

Il n'y a pas de bonne application sans bonne interface utilisateur : c'est la condition de son succès auprès des utilisateurs.

Les interfaces graphiques prennent une place de plus en plus importante dans le choix des applications par les utilisateurs, tant dans l'implémentation de concepts innovants qu'au niveau de l'ergonomie.

Comme sur bien des plates-formes, les interfaces d'applications Android sont organisées en vues et gabarits, avec néanmoins quelques spécificités.

Le concept d'interface

Une interface n'est pas une image statique mais un ensemble de composants graphiques, qui peuvent être des boutons, du texte, mais aussi des groupements d'autres composants graphiques, pour lesquels nous pouvons définir des attributs communs (taille, couleur, positionnement, etc.). Ainsi, l'écran ci-après (figure 4-1) peut-il être vu par le développeur comme un assemblage (figure 4-2).

La représentation effective par le développeur de l'ensemble des composants graphiques se fait sous forme d'arbre, en une structure hiérarchique (figure 4-3). Il peut n'y avoir qu'un composant, comme des milliers, selon l'interface que vous souhaitez représenter. Dans l'arbre ci-après (figure 4-3), par exemple, les composants ont été organisés en 3 parties (*haut, milieu et bas*).

Figure 4–1
Exemple d'un écran

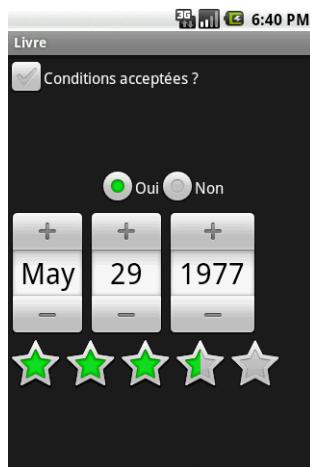


Figure 4–2
Le même écran vu
par le développeur

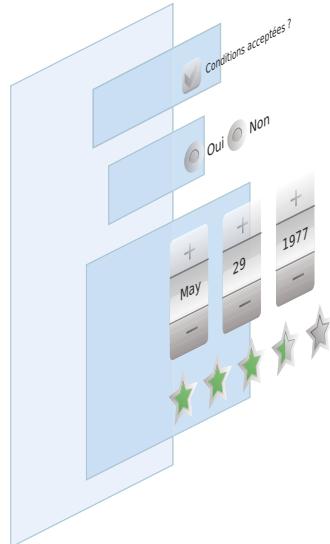
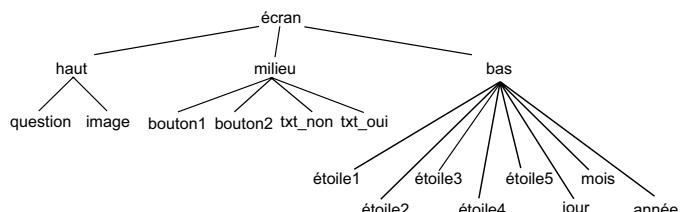


Figure 4–3
Arborescence
de l'interface
précédente



Cet exemple donne l'idée générale de l'organisation des interfaces – car les données graphiques ne sont pas exactement représentées ainsi. Nous verrons plus loin comment cela est géré pour Android.

Sous Android, vous pouvez décrire vos interfaces utilisateur de deux façons différentes (nous reviendrons plus tard en détail sur ce point) : avec une description déclarative XML ou directement dans le code d'une activité en utilisant les classes adéquates. La façon la plus simple de réaliser une interface est d'utiliser la méthode déclarative XML via la création d'un fichier XML que vous placerez dans le dossier `/res/layout` de votre projet.

Les vues

Le composant graphique élémentaire de la plate-forme Android est la *vue* : tous les composants graphiques (boutons, images, cases à cocher, etc.) d'Android héritent de la classe `View`.

Tout comme nous l'avions fait dans l'exemple précédent, Android vous offre la possibilité de regrouper plusieurs vues dans une structure arborescente à l'aide de la classe `ViewGroup`. Cette structure peut à son tour regrouper d'autres éléments de la classe `ViewGroup` et être ainsi constituée de plusieurs niveaux d'arborescence.

L'utilisation et le positionnement des vues dans une activité se fera la plupart du temps en utilisant une mise en page qui sera composée par un ou plusieurs gabarits de vues.

Positionner les vues avec les gabarits

Un gabarit, ou *layout* dans la documentation officielle, ou encore mise en page, est une extension de la classe `ViewGroup`. Il s'agit en fait d'un conteneur qui aide à positionner les objets, qu'il s'agisse de vues ou d'autres gabarits au sein de votre interface.

Vous pouvez imbriquer des gabarits les uns dans les autres, ce qui vous permettra de créer des mises en forme évoluées.

Comme nous l'avons dit plus haut, vous pouvez décrire vos interfaces utilisateur soit par une déclaration XML, soit directement dans le code d'une activité en utilisant les classes adéquates. Dans les deux cas, vous pouvez utiliser différents types de gabarits. En fonction du type choisi, les vues et les gabarits seront disposés différemment :

LinearLayout : permet d'aligner de gauche à droite ou de haut en bas les éléments qui y seront incorporés. En modifiant la propriété `orientation` vous pourrez signaler au gabarit dans quel sens afficher ses enfants : avec la valeur `horizontal`, l'affichage sera de gauche à droite alors que la valeur `vertical` affichera de haut en bas ;

RelativeLayout : ses enfants sont positionnés les uns par rapport aux autres, le premier enfant servant de référence aux autres ;

FrameLayout : c'est le plus basique des gabarits. Chaque enfant est positionné dans le coin en haut à gauche de l'écran et affiché par-dessus les enfants précédents, les cachant en partie ou complètement. Ce gabarit est principalement utilisé pour l'affichage d'un élément (par exemple, un cadre dans lequel on veut charger des images) ;

TableLayout : permet de positionner vos vues en lignes et colonnes à l'instar d'un tableau.

Voici un exemple de définition déclarative en XML d'une interface contenant un gabarit linéaire (le gabarit le plus commun dans les interfaces Android).

Code 4-1 : Interface avec un gabarit LinearLayout

```
<!-- Mon premier gabarit -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
</LinearLayout>
```

Chaque gabarit possède des attributs spécifiques, et d'autres communs à tous les types de gabarits. Parmi les propriétés communes, vous trouverez les propriétés `layout_weight` et `layout_height`. Celles-ci permettent de spécifier le comportement du remplissage en largeur et en hauteur des gabarits et peuvent contenir une taille (en pixels ou dpi) ou les valeurs constantes suivantes : `fill_parent` et `wrap_content`.

La valeur `fill_parent` spécifie que le gabarit doit prendre toute la place disponible sur la largeur/hauteur. Par exemple, si le gabarit est inclus dans un autre gabarit – parent (l'écran étant lui-même un gabarit) – et si le gabarit parent possède une largeur de 100 pixels, le gabarit enfant aura donc une largeur de 100 pixels. Si ce gabarit est le gabarit de base – le plus haut parent – de notre écran, alors ce dernier prend toute la largeur de l'écran.

Si vous souhaitez afficher le gabarit tel quel, vous pouvez spécifier la valeur `wrap_content`. Dans ce cas, le gabarit ne prendra que la place qui lui est nécessaire en largeur/hauteur.

À RETENIR Les unités de mesure

Pour spécifier les dimensions des propriétés de taille de vos composants graphiques (largeur, hauteur, marges, etc), vous pouvez spécifier plusieurs types d'unité. Le choix de l'utilisation d'une unité par rapport à une autre se fera en fonction de vos habitudes et si vous souhaitez spécifier une taille spécifique ou relative, par exemple : 10 px, 5 mm, 20 dp, 10 sp.

Voici les unités prises en charge par Android :

- pixel (px) : correspond à un pixel de l'écran ;
- pouce (in) : unité de longueur, correspondant à 2,54 cm. Basé sur la taille physique de l'écran ;
- millimètre (mm) : basé sur la taille physique de l'écran ;
- point (pt) : 1/72 d'un pouce ;
- pixel à densité indépendante (dp ou dip) : une unité relative se basant sur une taille physique de l'écran de 160 dpi. Avec cette unité, 1 dp est égal à 1 pixel sur un écran de 160 pixels. Si la taille de l'écran est différente de 160 pixels, les vues s'adapteront selon le ratio entre la taille en pixels de l'écran de l'utilisateur et la référence des 160 pixels ;
- pixel à taille indépendante (sp) : fonctionne de la même manière que les pixels à densité indépendante à l'exception qu'ils sont aussi fonction de la taille de polices spécifiée par l'utilisateur. Il est recommandé d'utiliser cette unité lorsque vous spécifiez les tailles des polices.

Ces deux dernières unités sont à privilégier car elles permettent de s'adapter plus aisément à différentes tailles d'écran et rendent ainsi vos applications plus portables. Notez que ces unités sont basées sur la taille physique de l'écran : l'écran ne pouvant afficher une longueur plus petite que le pixel, ces unités seront toujours rapportées aux pixels lors de l'affichage (1 cm peut ne pas faire 1 cm sur l'écran selon la définition de ce dernier).

Vous pouvez spécifier une taille précise en px (pixel) ou dip (dpi). Notez cependant que si vous avez besoin de spécifier une taille, notamment si vous avez besoin d'intégrer une image avec une taille précise, préférez les valeurs en *dip* à celles en *px*. En effet, depuis la version 1.6, Android est devenu capable de s'exécuter sur plusieurs tailles d'écrans. Les valeurs en *dip* permettent un ajustement automatique de vos éléments alors que les valeurs en *px* prennent le même nombre de pixels quelle que soit la taille de l'écran, ce qui peut très vite compliquer la gestion de l'affichage sur la multitude d'écrans disponibles.

Créer une interface utilisateur

La création d'une interface se traduit par la création de deux éléments :

- une définition de l'interface utilisateur (gabarits, etc.) de façon déclarative dans un fichier XML ;
- une définition de la logique utilisateur (comportement de l'interface) dans une classe d'activité.

Cela permet d'avoir une séparation stricte entre la présentation et la logique fonctionnelle de votre application. De plus, un intégrateur graphique pourra modifier l'interface sans interférer avec le code du développeur.

Définir votre interface en XML

Une bonne pratique est de définir intégralement votre interface dans la déclaration XML. Retenez néanmoins qu'il est aussi possible (et bon nombre de scénarios ne pourront s'en passer) d'instancier dynamiquement des vues depuis votre code.

Les fichiers de définition d'interface XML sont enregistrés dans le dossier `/layout` de votre projet. Prenez garde à ce que le nom du fichier ne comporte que des lettres minuscules et des chiffres .

Chaque fichier de définition d'interface, pour peu qu'il se trouve bien dans le répertoire `res/layout` de votre projet, possède un identifiant unique généré automatiquement par l'environnement de développement. De cette façon, vous pouvez y faire référence directement dans votre code. Par exemple, si le fichier se nomme `monLayout`, vous pouvez y faire référence dans votre code grâce à la constante `R.layout.monLayout`. Comme vous le verrez, l'utilisation des identifiants est très courante : ils vous serviront à récupérer des éléments, à spécifier des propriétés et à utiliser les nombreuses méthodes des activités (`findViewById`, `setContentView`, etc.).

Dans votre projet Android, créez un fichier `main.xml` dans le dossier `/layout` et placez-y le contenu suivant.

Code4-2 : Définition d'interface

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/monText"
    />
</LinearLayout>
```

Cette interface définit un gabarit de type `LinearLayout` et un composant graphique de saisie de texte de type `TextView`. Vous remarquerez que nous avons défini un attribut `android:id` avec la valeur `@+id/monText`. Cette valeur nous permettra de faire référence à cet élément dans le code avec la constante `R.id.monText`. Pour le moment nous n'allons pas détailler les éléments et les attributs : nous y reviendrons plus loin dans ce chapitre.

Le complément ADT génère automatiquement un identifiant pour cette définition d'interface. Nommée `main.xml` et placée dans le répertoire `res/layout`, cet identifiant sera `R.layout.main`.

Associer votre interface à une activité et définir la logique utilisateur

Dans une application, une interface est affichée par l'intermédiaire d'une activité. Vous devez donc avant toute chose créer une activité en ajoutant une nouvelle classe à votre projet dérivant de la classe `Activity`.

Le chargement du contenu de l'interface s'effectue à l'instanciation de l'activité. Redéfinissez la méthode `onCreate` de l'activité pour y spécifier la définition de l'interface à afficher via la méthode `setContentView`. Cette méthode prend en paramètre un identifiant qui spécifie quelle ressource de type interface doit être chargée et affichée comme contenu graphique. Nous utilisons l'identifiant généré automatiquement pour spécifier l'interface que nous avons précédemment créée.

Code 4-3 : Spécifier une vue à l'activité

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

public class Main extends Activity {

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

L'interface (code 4-2) a défini un composant graphique `TextView` vide. Pour y accéder depuis le code et pouvoir le manipuler, vous devez d'abord récupérer une instance de l'objet avec la méthode `findViewById` de l'activité. Une fois l'objet récupéré, vous pourrez le manipuler de la même façon que n'importe quel objet, par exemple pour modifier son texte. L'affichage répercutera le changement.

Le code suivant récupère le composant graphique `TextView` que nous avons nommé `monText` dans l'interface, puis utilise la méthode `setText` pour changer le contenu de son texte.

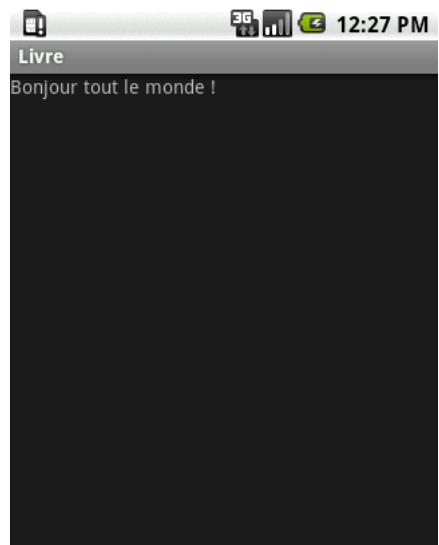
Code 3-4 : Récupérer un élément de l'interface et interagir avec

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class Main extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        TextView monTexte = (TextView)findViewById(R.id.monText);
        monTexte.setText("Bonjour tout le monde !");
    }
}
```

Le résultat dans l'émulateur Android est celui de la figure 3-5.

Figure 4–5
Le résultat de l'exemple 3-4





le paramètre *gravity*

- Dans certains cas, vous serez amené à positionner des vues plutôt que de les aligner.
- Pour se faire, il faudra utiliser le paramètre **gravity**,
- le paramètre **gravity** spécifie l'attrait d'un composant vers un coin de l'écran.



le paramètre *gravity*

- Nous allons maintenant modifier **la gravité** de deux éléments **TextView** dans la définition de l'interface afin de les aligner en **haut** et en **bas** de notre gabarit.
- Vous noterez l'utilisation du caractère | pour spécifier une combinaison de valeurs et placer vos éléments au plus proche de vos besoins.

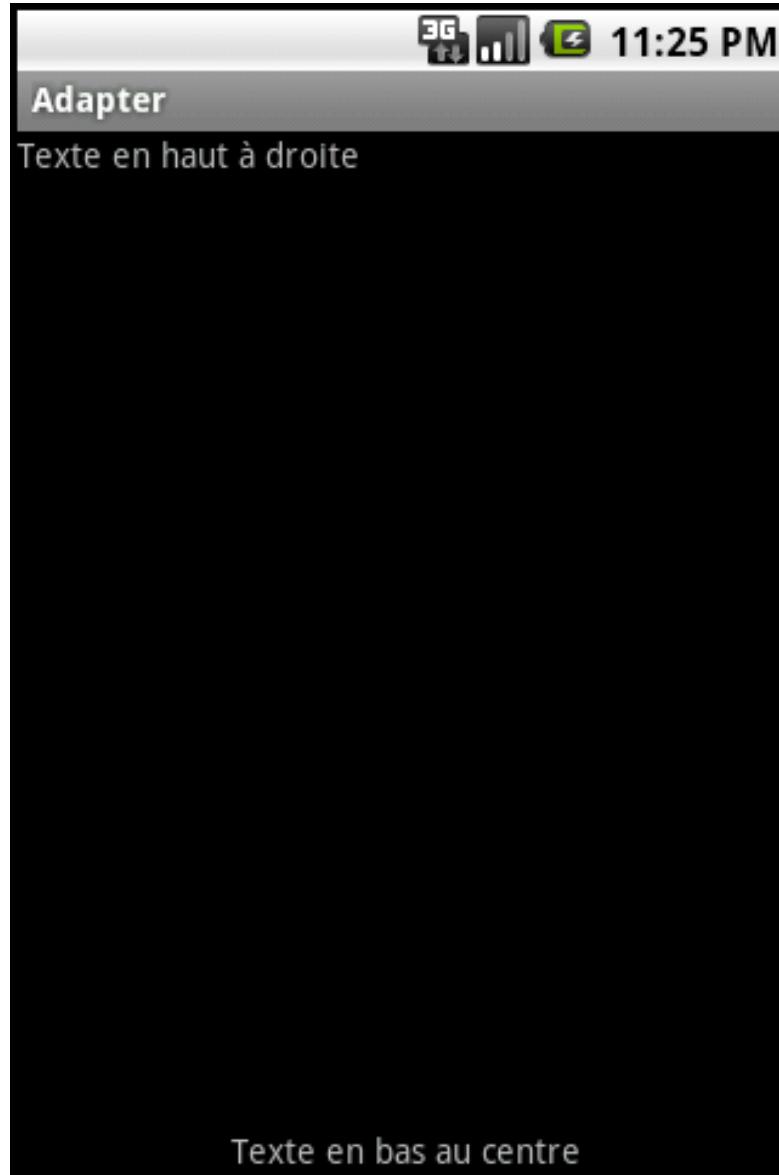


le paramètre gravity

```
<?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"
    >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/monText"
        android:text="Texte en haut à droite"
        android:gravity="top|right"
        />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="bottom|center_horizontal"
        android:id="@+id/monText2"
        android:text="Texte en bas au centre"
        />
</LinearLayout>
```



le paramètre gravity





le paramètre gravity

- Dans cet exemple, la gravité top | right (haut | droite) est appliquée sur l'élément `TextView` (*texte en haut à droite*) qui occupe toute la largeur grâce à la valeur `fill_parent` du paramètre `layout_width`. Ce paramétrage indique au texte de se positionner le plus en haut et à droite de l'espace qu'il occupe.

- Si à la place de la valeur `fill_parent` nous avions spécifié `wrap_content`, le texte aurait été aligné visuellement à gauche car la place qu'occuperait l'élément `TextView` serait limitée à son contenu.



le paramètre *gravity*

- Pour aligner en bas le deuxième `TextView`, nous avons dû paramétrer la propriété `layout_width` et `layout_height` avec la valeur `fill_parent` et spécifier la valeur `bottom | center_horizontal` (bas | centre horizontal) à la propriété `gravity`.



Intégrer une image dans votre interface

- Pour ajouter une image dans votre interface, utilisez la vue de type **ImageView**.
- Vous pouvez spécifier la source de l'image directement dans votre définition XML, via l'attribut **src**,
 - ou alors utiliser la méthode **setImageResource** en passant **l'identifiant** en paramètre.
- Pour l'exemple suivant, nous utilisons une image nommée **icon.png** qui se trouve déjà dans les ressources de votre projet (cette ressource est incluse dans le modèle de projet Android d'Eclipse et se trouve dans le dossier **/res/drawable**).



Intégrer une image dans votre interface

- Pour spécifier la taille d'une image dans la valeur de la propriété **src**, plusieurs options s'offrent à vous :
 - soit vous utilisez la dimension réelle de l'image (**wrap_content**)
 - ou la taille de l'écran (**fill_parent**),
 - soit vous spécifiez la taille exacte (exemple : **100 px, 25 dp**, etc).

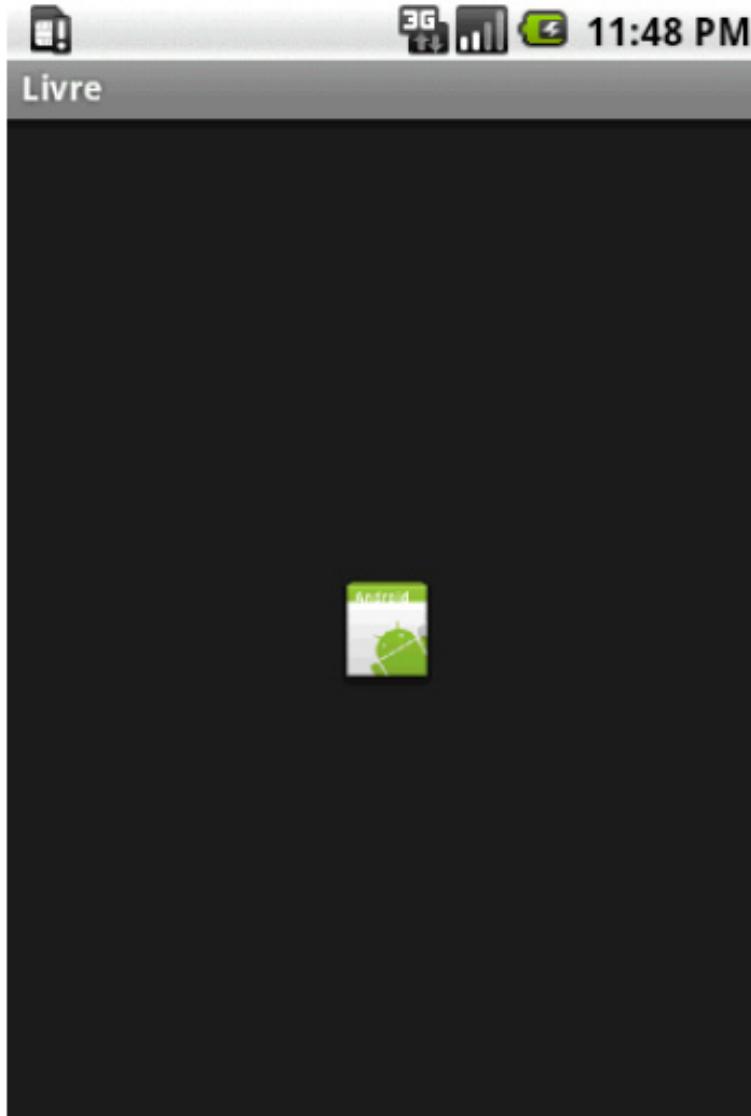


Intégrer une vue image dans une interface

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:gravity="center_vertical|center_horizontal"
    >
    <ImageView
        android:id="@+id/monImage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/icon"
        >
    </ImageView>
</LinearLayout>
```



Intégrer une vue image dans une interface





Intégrer une vue image dans une interface

```
ImageView monImage = ...  
monImage.setImageResource(R.drawable.icon);
```



Intégrer une boîte de saisie de texte

- Pour proposer à l'utilisateur de saisir du texte, vous pouvez utiliser la vue **EditText**.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <!--
        L'élément EditText permettra à
        l'utilisateur de saisir son nom.
    -->
    <EditText
        android:id="@+id/monEditText"
        android:layout_height="wrap_content"
        android:hint="Taper votre nom"
        android:layout_width="fill_parent">
</EditText>
```

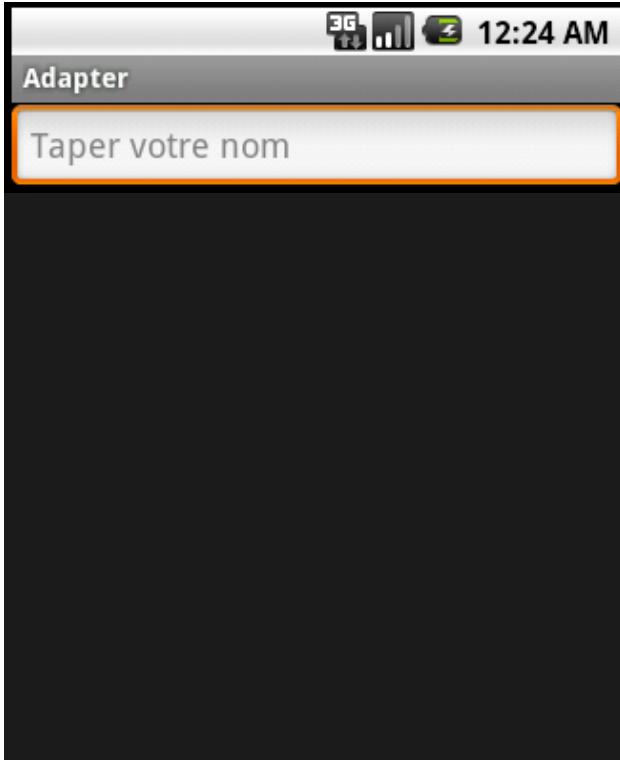


Ajout d'une vue de saisie de texte à l'interface utilisateur

- La propriété **hint** de l'élément **EditText** permet l'affichage en **fond** d'une aide.
- Tant qu'aucun texte n'est saisi dans la zone dédiée, cette aide apparaît **grisée** sur le fond.
- Cela peut vous éviter de mettre un texte de description avant ou au-dessus de chacune des zones de saisie et d'indiquer à l'utilisateur **l'objet de sa saisie**.



Ajout d'une vue de saisie de texte à l'interface utilisateur



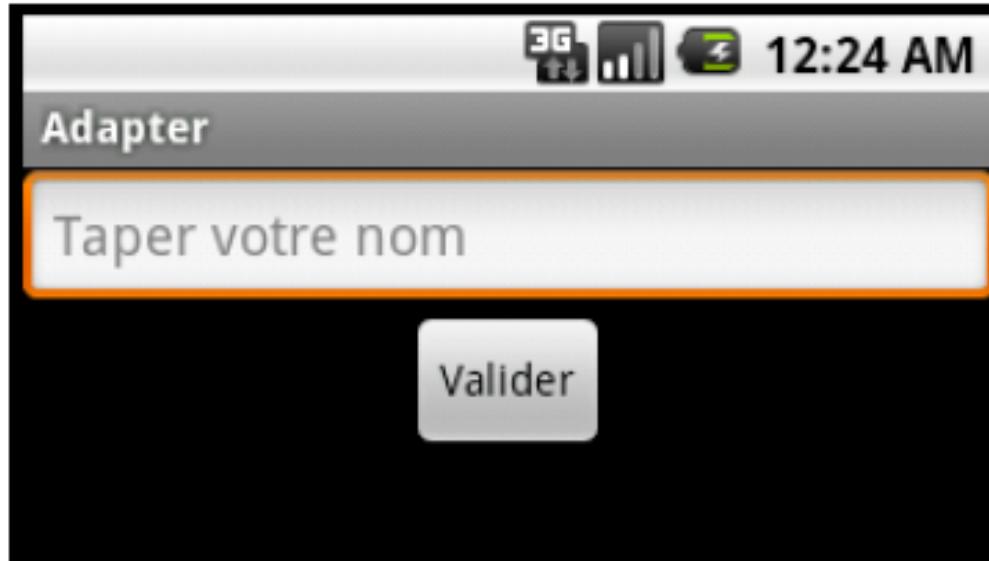


Bouton de validation de saisie

```
<!--  
    Le Bouton qui permettra à l'utilisateur  
    de valider sa saisie  
-->  
<Button  
    android:id="@+id/monBouton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Valider"  
    android:layout_gravity="center_horizontal">  
</Button>  
</LinearLayout>
```



Bouton de validation de saisie



- Voyons maintenant comment récupérer ce que l'utilisateur a inscrit lorsqu'il clique sur le bouton Valider.

- Modifiez la méthode **onCreate** de l'activité Main de votre projet.



Récupérer la saisie de l'utilisateur

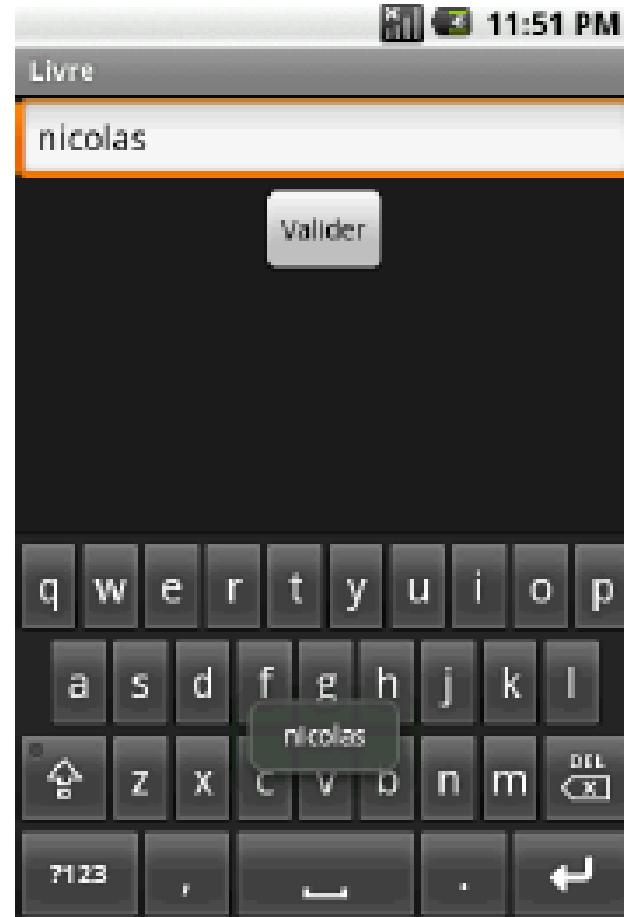
...

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
  
    // nous ajoutons un écouteur de type OnClickListener sur le bouton  
    // de validation.  
    ((Button)findViewById(R.id.monBouton)).setOnClickListener(  
        ↗ new OnClickListener() {  
  
        @Override  
        public void onClick(View v) {  
  
            // On récupère notre EditText  
            EditText texte = ((EditText)findViewById(R.id.monEditText));  
            // On garde la chaîne de caractères  
            String nom = texte.getText().toString();  
            // On affiche ce qui a été tapé  
            Toast.makeText(Main.this, nom, Toast.LENGTH_SHORT).show();  
        }  
    });  
}
```

...



Récupérer la saisie de l'utilisateur





Intégrer d'autres composants graphiques

- Nous allons maintenant intégrer dans notre interface plusieurs types de vues que vous serez amené à utiliser dans vos applications :
- une case à cocher (**CheckBox**),
- un bouton image (**ImageButton**),
- deux boutons radio (**RadioGroup** et **RadioButton**),
- un contrôle de saisie de date (**DatePicker**),
-



Diverses vues dans un même gabarit

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <!-- Case à cocher -->
    <CheckBox
        android:id="@+id/CheckBox01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Conditions acceptées ?">
    </CheckBox>
    <!-- Bouton avec une Image -->
    <ImageButton
        android:id="@+id/ImageButton01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/icon">
        <!-- @drawable/icon est une Image qui se trouve dans le dossier /res/
        drawable de notre projet -->
    </ImageButton>
```



Diverses vues dans un même gabarit

```
<!-- Groupe de boutons radio -->
<RadioGroup
    android:id="@+id/RadioGroup01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_gravity="center_horizontal">
    <!-- Radio Bouton 1 -->
    <RadioButton
        android:id="@+id/RadioButton01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Oui"
        android:checked="true">

        <!-- Nous avons mis checked=true par défaut sur notre 1er bouton
radio afin qu'il soit coché -->
    </RadioButton>
    <!-- Bouton radio 2 -->
    <RadioButton
        android:id="@+id/RadioButton02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Non">
    </RadioButton>
</RadioGroup>
```

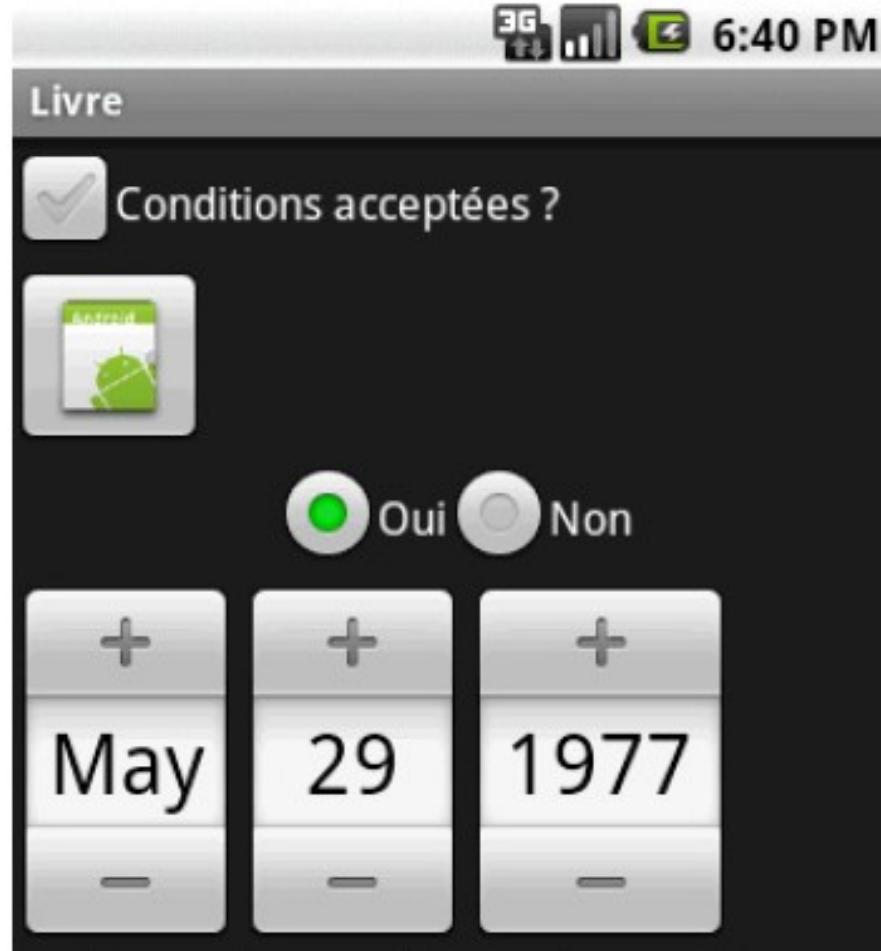


Diverses vues dans un même gabarit

```
<!-- Sélectionneur de date -->
<DatePicker
    android:id="@+id/DatePicker01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
</DatePicker>
```



Diverses vues dans un même gabarit



De la même façon, modifiez votre fichier `main.java` pour y placer le code suivant.

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.CheckBox;
import android.widget.DatePicker;
import android.widget.ImageButton;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

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

        // On récupère notre case à cocher pour intercepter l'événement
        // d'état (cochée ou pas)
        ((CheckBox)findViewById(R.id.CheckBox01)).setOnCheckedChangeListener(
            ↪ new CheckBox.OnCheckedChangeListener() {
                public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                    afficheToast("Case cochée ? : " + ((isChecked)?"Oui":"Non"));
                }
            });

        // On récupère notre sélectionneur de date (DatePicker) pour attraper
        // l'événement du changement de date
        // Attention, le numéro de mois commence à 0 dans Android, mais pas les jours.
        // Donc si vous voulez mettre le mois de Mai, vous devrez fournir 4 et non 5
        ((DatePicker)findViewById(R.id.DatePicker01)).init(1977, 4, 29,
            ↪ new DatePicker.OnDateChangedListener() {

    @Override
    public void onDateChanged(DatePicker view, int year, int monthOfYear,
        ↪ int dayOfMonth) {
        // On affiche la nouvelle date qui vient d'être changée dans notre
        // DatePicker
        afficheToast("La date a changé\nAnnée : " + year + " | Mois : "
            ↪ + monthOfYear + " | Jour : " + dayOfMonth);
    }
});
```

```

// On récupère notre groupe de bouton radio pour attraper le choix de
// l'utilisateur
((RadioGroup)findViewById(R.id.RadioGroup01)).setOnCheckedChangeListener(
    ↪ new RadioGroup.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(RadioGroup group, int checkedId) {
            // On affiche le choix de l'utilisateur
            afficheToast("Vous avez répondu : "
                ↪ + ((RadioButton)findViewById(checkedId)).getText());
        }
    });

// On récupère notre Bouton Image pour attraper le clic effectué par
// l'utilisateur
((ImageButton)findViewById(R.id.ImageButton01)).setOnClickListener(
    ↪ new OnClickListener() {
        @Override
        public void onClick(View v) {
            // On affiche un message pour signaler que le bouton image a été pressé
            afficheToast("Bouton Image pressé");
        }
    });
}

// Méthode d'aide qui simplifiera la tâche d'affichage d'un message
public void afficheToast(String text)
{
    Toast.makeText(this, text, Toast.LENGTH_SHORT).show();
}
}

```

Dans cet exemple, nous employons diverses vues que vous serez souvent amené à utiliser :

- la case à cocher ([CheckBox](#)) : propose un choix basique à vos utilisateurs : « oui » ou « non ». Pour récupérer à tout moment l'état de la case à cocher, il vous suffit de le récupérer avec la méthode `isChecked`. Dans notre exemple nous récupérons l'état dès qu'il change, grâce à l'écouteur [OnCheckedChangeListener](#) que nous

avons associé à la case à cocher. Ceci nous permet de récupérer immédiatement le changement d'état et si nécessaire d'exécuter du code en conséquence ;

- le bouton image (`ImageButton`) : fonctionne de la même façon qu'un bouton classique mais présenté précédemment sous la forme de l'image choisie. La principale différence avec un bouton classique est que l'on ne peut ajouter un texte à afficher : seule l'image est affichée ;
- le groupe de boutons radio et les boutons radio (`RadioGroup` et `RadioButton`) : propose une liste de choix à réponse unique (seul un des boutons du groupe de boutons pourra être coché à un instant t). Afin que les boutons radio (de type `RadioButton`) puissent basculer de l'un à l'autre, vous devez les regrouper dans un élément `ViewGroup` de type `RadioGroup`. Si vos boutons radio ne sont pas regroupés dans un `RadioGroup`, ils seront tous indépendants. Dans notre exemple, nous attrapons le changement d'état des boutons radio dès qu'ils sont pressés grâce à l'écouteur `OnCheckedChangeListener` que nous avons associé au `RadioGroup`. Si vous souhaitez récupérer le bouton radio pressé à n'importe quel moment il vous suffira d'appeler la méthode `getCheckedRadioButtonId` ;
- le sélectionneur de date (`DatePicker`) : affiche les boutons nécessaires pour saisir une date. Attention celui-ci s'adapte visuellement à la langue et région du téléphone. Ainsi, un téléphone en langue « Anglais US » affiche la date sous la forme Mois-Jour-Année tandis que le même téléphone en langue française l'affiche sous la forme Jour-Mois-Année. Dans notre exemple, nous récupérons le changement de date de notre sélectionneur de date dès que l'utilisateur modifie une valeur grâce à l'écouteur `OnDateChangedListener` que nous lui avons associé. Pour récupérer les valeurs sélectionnées, vous pouvez à tout moment utiliser les méthodes `getDayOfMonth`, `getMonth`, `getYear`. Attention, comme signalé en commentaire dans l'exemple, le numéro des mois commence à 0 (janvier = 0, février = 1 etc.) ce qui n'est pas le cas pour les jours et les années ;



Créer un menu pour une activité



Créer un menu pour une activité

- Tous les modèles Android possèdent un **bouton Menu**. Grâce à ce bouton, il vous est possible de proposer à vos utilisateurs des **fonctionnalités supplémentaires n'apparaissant pas par défaut à l'écran** et d'ainsi mieux gérer la taille limitée de l'écran d'un appareil mobile.

- Chaque menu est propre à une activité, c'est pourquoi toutes les opérations que nous allons traiter dans cette partie se réfèrent à une activité. Pour proposer plusieurs menus vous aurez donc besoin de le faire dans chaque activité de votre application.



Création d'un menu

- Pour créer un menu il vous suffit de surcharger la méthode **onCreateOptionsMenu** de la classe Activity.
- Cette méthode est appelée la première fois que l'utilisateur appuie sur **le bouton menu** de son téléphone.
- Elle reçoit en paramètre **un objet de type Menu** dans lequel nous ajouterons nos éléments ultérieurement.



Création d'un menu

- Si l'utilisateur appuie une seconde fois sur le bouton Menu, cette méthode ne sera plus appelée tant que l'activité ne sera pas **détruite puis recréée**.

- Si vous avez besoin d'ajouter, de supprimer ou de modifier un élément du menu après coup, il vous faudra surcharger une autre méthode, la méthode **onPrepareOptionsMenu** que nous aborderons plus tard dans cette partie.



Création d'un menu

- Pour créer un menu, commencez par créer un fichier de définition d'interface nommé **main.xml**:

Code 5-18 : Définition de l'interface de l'exemple avec menu

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <TextView
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Appuyez sur la touche menu">
    </TextView>
</LinearLayout>
```

Code 5-19 : Création d'un menu

```
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class Main extends Activity {

    // Pour faciliter la gestion des menus
    // nous créons des constantes
    private final static int MENU_PARAMETRE = 1;
    private final static int MENU_QUITTER = 2;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Nous créons un premier menu pour accéder aux paramètres.
        // Pour ce premier élément du menu, nous l'agrémentons
        // d'une image.
        menu.add(0, MENU_PARAMETRE, Menu.NONE, "Paramètres").setIcon(R.drawable.icon);
        // Nous créons un second élément.
        // Celui-ci ne comportera que du texte.
        menu.add(0, MENU_QUITTER, Menu.NONE, "Quitter");

        return true;
    }
}
```



Création d'un menu

- menu.add(groupId, itemId, ordre, "nom du menu") ;**
- Cette méthode permet de créer un nouveau menu. Les paramètres nécessaires sont les suivants :
 - **groupId** : identifiant du groupe. Il est possible de regrouper les éléments,
 - **itemId** : identifiant de ce menu. Il nous sera utilisé pour identifier ce menu parmi les autres. On doit donner un identifiant différent à chaque menu (1, 2, 3... par exemple).
 - **ordre** : associer un ordre d'affichage au menu. On donnera ici la valeur 0.
 - Nom du menu : chaîne qui représente le titre du menu.



Création d'un menu

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case MENU_PARAMETRE:  
            // Lorsque l'utilisateur cliquera sur le menu 'paramètres',  
            // un message s'affichera.  
            Toast.makeText(Main.this, "Ouverture des paramètres",  
                    Toast.LENGTH_SHORT).show();  
            // onOptionsItemSelected retourne un booléen,  
            // nous lui envoyons la valeur "true" pour signaler  
            // que tout s'est correctement déroulé.  
            return true;  
  
        case MENU_QUITTER:  
            // Lorsque l'utilisateur cliquera sur le menu 'Quitter',  
            // nous fermerons l'activité avec la méthode finish().  
            finish();  
            return true;  
        default:  
            return true;  
    }  
}
```

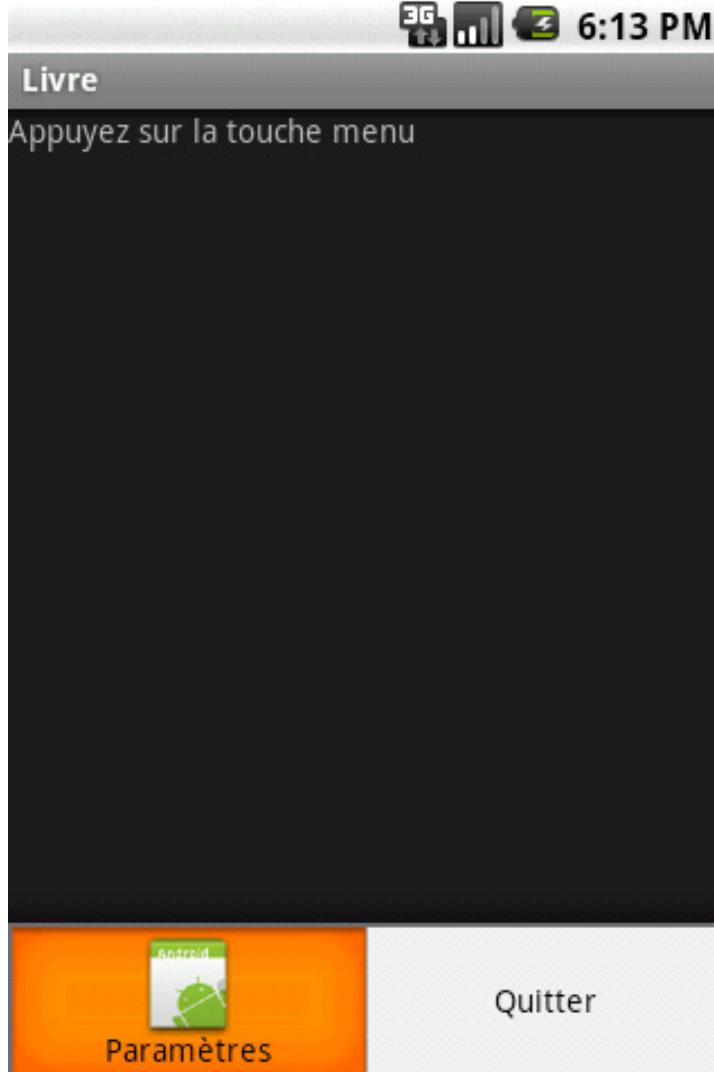


Création d'un menu

- Dans le code précédent, nous avons surchargé deux méthodes, l'une responsable de la création du menu de l'activité, l'autre de la logique lors de la sélection par l'utilisateur d'un élément du menu.
- La méthode **onCreateOptionsMenu** sera appelée uniquement la première fois lorsque l'utilisateur appuiera sur le bouton **Menu**.
- Dans cette méthode, nous créons deux éléments de menu **Paramètres** et **Quitter**.
- La méthode **onOptionsItemSelected** est appelée lorsque l'utilisateur clique sur l'un des éléments du menu.
- Dans notre cas, si l'utilisateur clique sur l'élément **Paramètres** l'application affiche **un message**, alors qu'un clic sur **Quitter** ferme l'activité grâce à la méthode **finish**.



Création d'un menu





Mettre à jour dynamiquement un menu

- Il vous sera peut-être nécessaire de changer l'intitulé, de supprimer ou de rajouter un élément du menu en cours de fonctionnement.
- La méthode **onCreateOptionsMenu** n'étant appelée qu'une fois, la première fois où l'utilisateur clique sur le bouton Menu, vous ne pourrez pas mettre à jour votre menu dans cette méthode.
- Pour modifier le menu après coup, par exemple dans le cas où votre menu propose de s'authentifier et qu'une fois authentifié, vous souhaitez proposer à l'utilisateur de se déconnecter, vous devrez surcharger la méthode **onPrepareOptionsMenu**.



Mettre à jour dynamiquement un menu

- Un objet de type **Menu** est envoyé à cette méthode qui est appelée à chaque fois que l'utilisateur cliquera sur le bouton **Menu**.
- Modifiez le code 5-19 pour rajouter la redéfinition de la méthode **onPrepareOptionsMenu** :

Code 5-20 : Modification dynamique du menu

```
@Override  
public boolean onPrepareOptionsMenu(Menu menu) {  
    // Nous modifions l'intitulé de notre premier menu  
    // Pour l'exemple nous lui donnerons un intitulé  
    // différent à chaque fois que l'utilisateur cliquera  
    // sur le bouton menu à l'aide de l'heure du système  
    menu.getItem(0).setTitle(SystemClock.elapsedRealtime()++);  
  
    return super.onPrepareOptionsMenu(menu);  
}
```

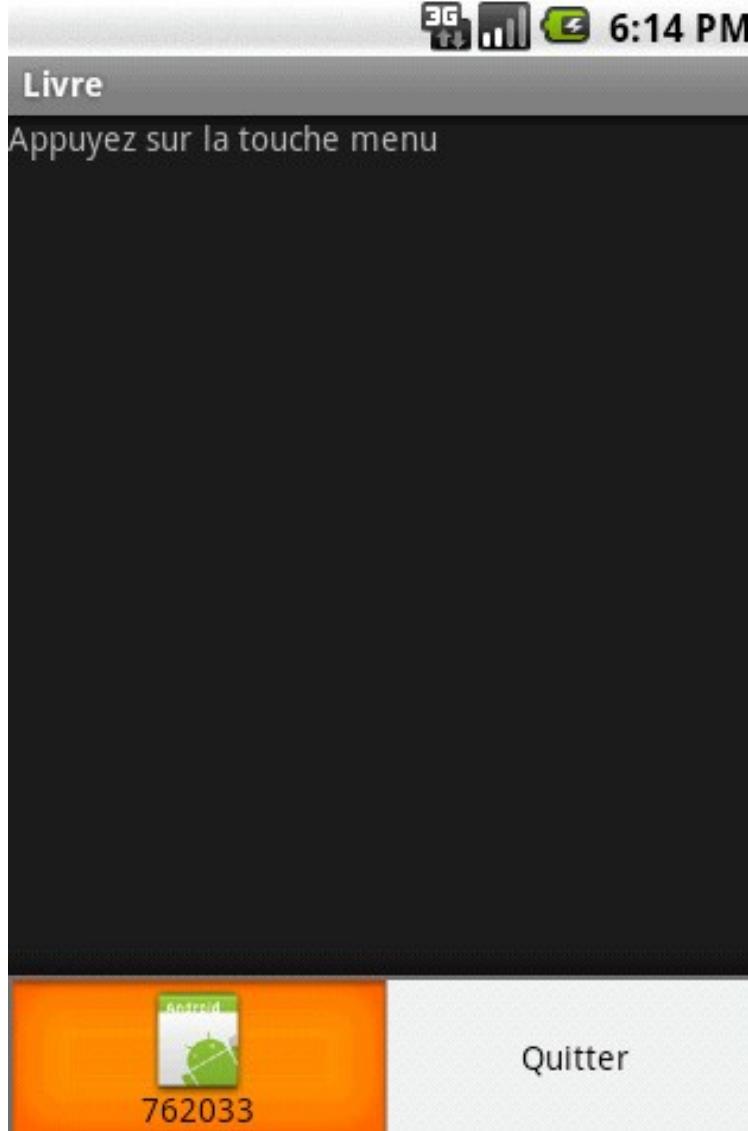


Mettre à jour dynamiquement un menu

- Dans cet exemple, lorsque l'utilisateur appuie sur le bouton **Menu**, l'intitulé du premier menu affiche une valeur différente à chaque fois.
- Vous pourrez placer à cet endroit le code nécessaire pour adapter l'intitulé que vous souhaitez afficher à l'utilisateur s'il est nécessaire de changer le menu.
- Si l'utilisateur appuie à nouveau, l'intitulé changera de concert. S'il rappuie encore, l'intitulé changera de même.

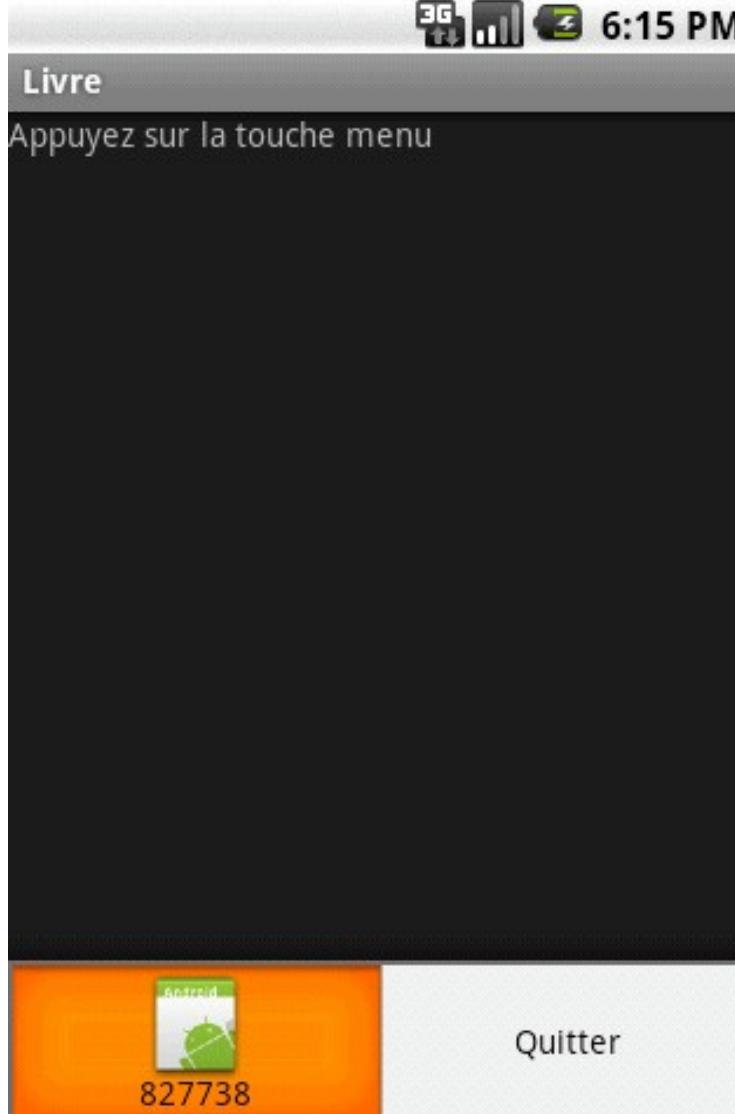


Mettre à jour dynamiquement un menu





Mettre à jour dynamiquement un menu





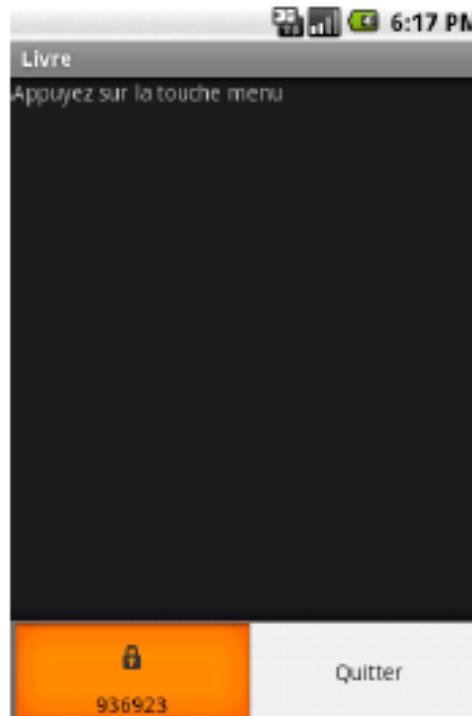
Mettre à jour dynamiquement un menu

- De la même façon, vous pouvez changer l'image du menu comme ceci :

Code 5-21 : Changer l'image d'un élément de menu

```
menu.getItem(0).setIcon(android.R.drawable.icon_secure);
```

Figure 5-15
L'icône d'un élément du menu
changée dynamiquement





Créer des sous-menus

- Créer des sous-menus peut se révéler utile pour proposer plus d'options sans encombrer l'interface utilisateur.
- Pour ajouter un sous-menu, vous devrez ajouter un menu de type **SubMenu** et des éléments le composant.
- Le paramétrage est le même que pour un menu, à l'exception de l'image : vous ne pourrez pas ajouter d'image sur les éléments de vos sous-menus.
- Néanmoins, il sera possible **d'ajouter une image dans l'entête du sous-menu**.
- Remplacez la méthode **onCreateOptionsMenu** du code 5-19 par celle-ci :



Créer des sous-menus

Code 5-22 : Création de sous-menus

```
private final static int SOUSMENU_VIDEO= 1000;
private final static int SOUSMENU_AUDIO= 1001;

...
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Nous créons un premier menu sous forme de sous-menu
    // Les sous menu associés à ce menu seront ajoutés à ce sous-menu
    SubMenu sousMenu = menu.addSubMenu(0, MENU_PARAMETRE, Menu.NONE,
                                        "Paramètres").setIcon(R.drawable.icon);
    // Nous ajoutons notre premier sous-menu
    sousMenu.add(0, SOUSMENU_AUDIO, Menu.NONE, "Audio").setIcon(R.drawable.icon);
    // Nous ajoutons notre deuxième sous-menu
    sousMenu.add(0, SOUSMENU_VIDEO, Menu.NONE, "Vidéo");

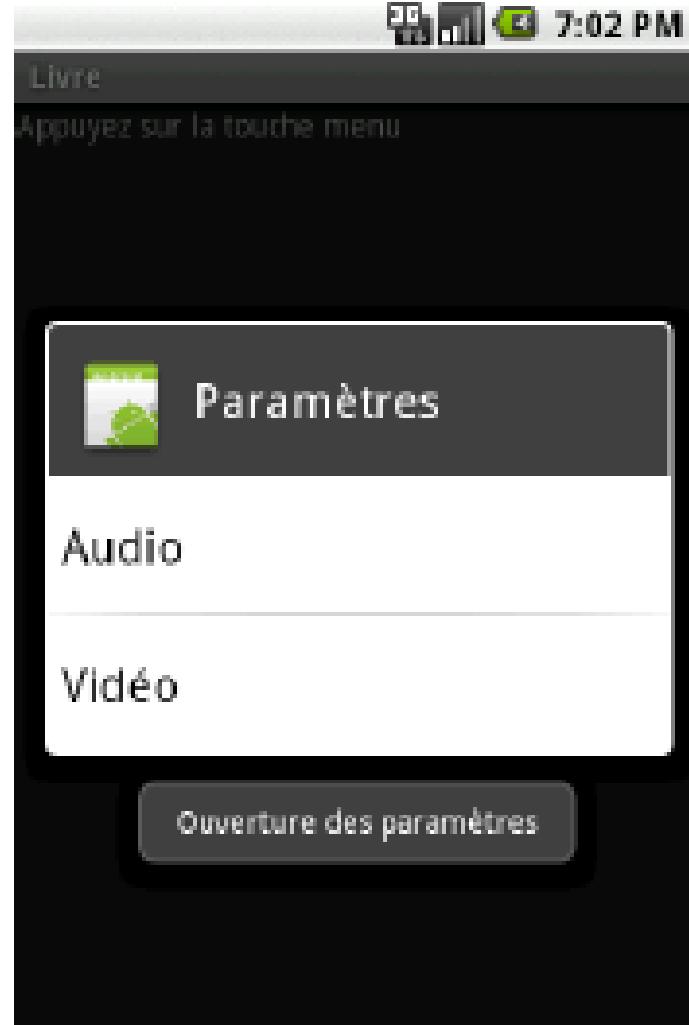
    // Il est possible de placer une
    // image dans l'entête du sous-menu
    // Il suffit de le paramétrier avec la méthode
    // setHeaderIcon de notre objet SubMenu
    sousMenu.setHeaderIcon(R.drawable.icon);

    // Nous créons notre deuxième menu
    menu.add(0, MENU_QUITTER, Menu.NONE, "Quitter");

    return true;
}
```



Créer des sous-menus





Créer des sous-menus

- Lorsque l'utilisateur clique sur le bouton Menu, le menu qui s'affiche est identique à l'exemple précédent.
- Néanmoins en cliquant sur le menu **Paramètres** un sous-menu s'affiche sous forme de liste.



Créer un menu contextuel



Créer un menu contextuel

- La plate-forme Android autorise également la création des menus contextuels, autrement dit de menus dont le contenu change en fonction du contexte.
- Sous Android, l'appel d'un menu contextuel se fait lorsque l'utilisateur effectue un clic de quelques secondes sur un élément d'interface graphique.
- Les menus contextuels fonctionnent d'ailleurs de façon similaire aux sous-menus.



Créer un menu contextuel

- La création d'un menu contextuel se fait en deux étapes :
 - tout d'abord la création du menu proprement dit, puis
 - son enregistrement auprès de la vue, avec la méthode **registerForContextMenu** ; en fournissant en paramètre la vue concernée.



Créer un menu contextuel

- Concrètement, redéfinissez la méthode **onCreateContextMenu** et ajoutez-y les menus à afficher en fonction de la vue sélectionnée par l'utilisateur.

- Pour sélectionner les éléments qui s'y trouveront, redéfinissez la méthode **onContextItemSelected**.



Créer un menu contextuel

- Créez un nouveau fichier de définition d'une interface comme ceci :

Code 5-23 : Définition de l'interface de l'exemple avec menu contextuel

```
<?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_vertical|center_horizontal">
    <TextView
        android:id="@+id/monTexteContextMenu"
        android:layout_height="wrap_content"
        android:text="Cliquez ici 3 sec"
        android:textSize="30dip"
        android:layout_width="fill_parent"
        android:gravity="center_horizontal">
    </TextView>
</LinearLayout>
```



Créer un menu contextuel

Code 5-24 : Crédit d'un menu contextuel

```
...
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.ContextMenu.ContextMenuItemInfo;
...

// Pour faciliter notre gestion des menus
// nous créons des variables de type int
// avec des noms explicites qui nous aideront
// à ne pas nous tromper de menu
private final static int MENU_CONTEXT_1= 1;
private final static int MENU_CONTEXT_2= 2;

@Override
public void onCreate(Bundle savedInstanceState) {
    ...

    // Nous enregistrons notre TextView pour qu'il réagisse
    // au menu contextuel
    registerForContextMenu((TextView) findViewById(R.id.monTexteContextMenu));
}
```



Créer un menu contextuel

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenuInfo menuInfo) {
    // Nous vérifions l'identifiant de la vue.
    // Si celle-ci correspond à une vue pour laquelle nous souhaitons
    // réagir au clic long, alors nous créons un menu.
    // Si vous avez plusieurs vues à traiter dans cette méthode,
    // il vous suffit d'ajouter le code nécessaire pour créer les
    // différents menus.
    switch (v.getId()) {
        case R.id.monTexteContextMenu:
            {
                menu.setHeaderTitle("Menu contextuel");
                menu.setHeaderIcon(R.drawable.icon);
                menu.add(0, MENU_CONTEXT_1, 0, "Mon menu contextuel 1");
                menu.add(0, MENU_CONTEXT_2, 0, "Mon menu contextuel 2");
                break;
            }
    }
    super.onCreateContextMenu(menu, v, menuInfo);
}
```

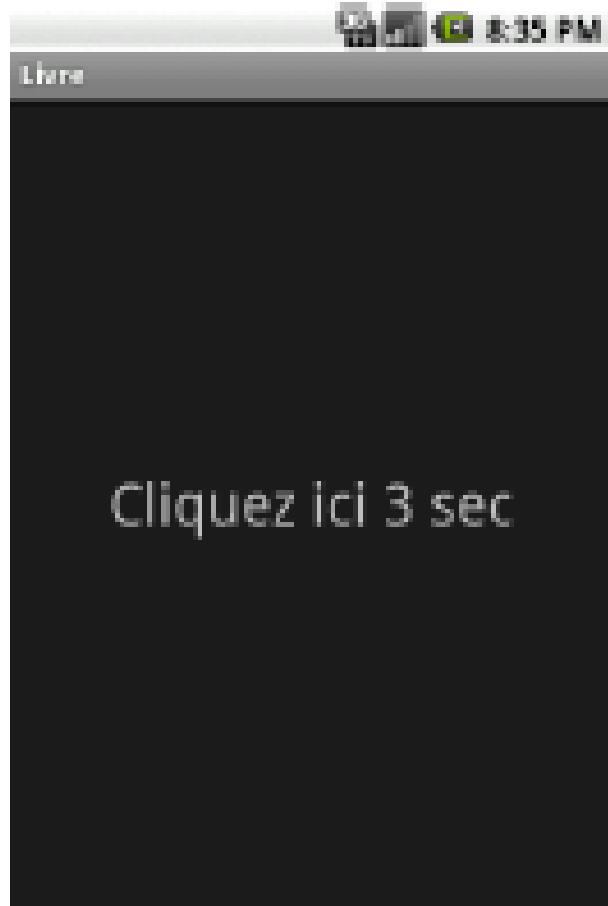


Créer un menu contextuel

```
@Override  
public boolean onContextItemSelected(MenuItem item) {  
    // Grâce à l'identifiant de l'élément  
    // sélectionné dans notre menu nous  
    // effectuons une action adaptée à ce choix  
    switch (item.getItemId()) {  
        case MENU_CONTEXT_1:  
            {  
                Toast.makeText(Main.this, "Menu contextuel 1 cliqué !",  
                    Toast.LENGTH_SHORT).show();  
                break;  
            }  
        case MENU_CONTEXT_2:  
            {  
                Toast.makeText(Main.this, "Menu contextuel 2 cliqué !",  
                    Toast.LENGTH_SHORT).show();  
                break;  
            }  
    }  
    return super.onContextItemSelected(item);  
}
```



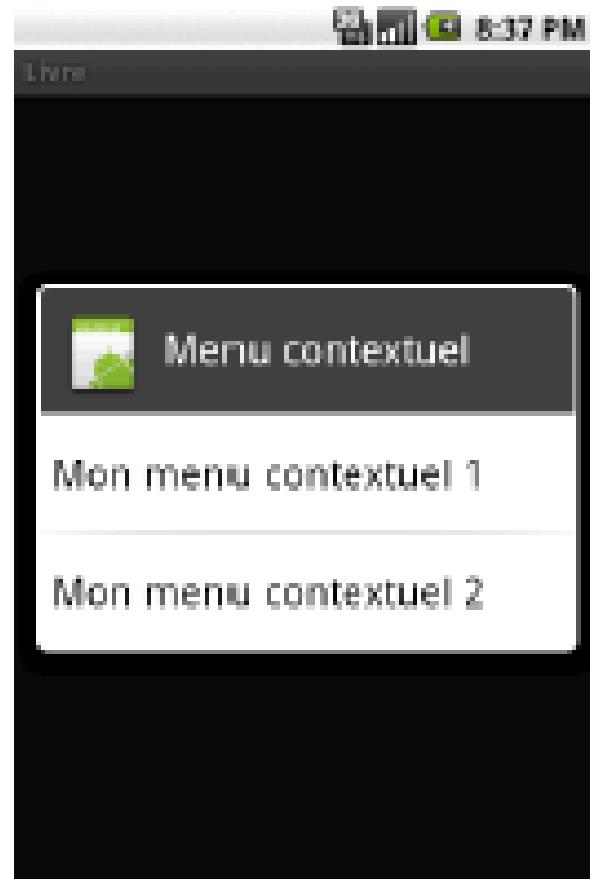
Créer un menu contextuel



- Si l'utilisateur clique pendant trois secondes sur la vue **TextView** centrale, le menu contextuel s'affiche.



Créer un menu contextuel





Créer un menu contextuel

Pour insérer une image dans le gabarit de mise en page de l'interface, ajoutez les lignes suivantes dans le fichier XML après la balise `<TextView>` :

Code 5-25 : Ajout d'une image dans le menu contextuel de l'exemple

```
...
</TextView>
<ImageView
    android:id="@+id/ImageView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/icon">
</ImageView>
...
```



Créer un menu contextuel

- À ce stade, si l'utilisateur clique quelques secondes sur l'image, il ne se produira rien car la vue de l'image n'a pas enregistré de menu contextuel.
- Vous devez ajouter la ligne suivante dans la méthode **onCreate** de votre fichier main.java :

Code 5-26 : Enregistrer le contrôle de l'image pour réagir au menu contextuel

```
registerForContextMenu((TextView) findViewById(R.id.monTexteContextMenu));  
...  
registerForContextMenu((ImageView) findViewById(R.id.monImageContext));
```



Créer un menu contextuel

- L'image est désormais enregistrée et un clic dessus déclenchera l'apparition du menu contextuel. Il reste à adapter les méthodes **onCreateContextMenu** et **onContextItemSelected** pour que les menus contextuels destinés à cette image s'affichent correctement :

Code 5-27 : Modification du code 5-24 pour gérer le menu contextuel de l'image

```
--  
private final static int MENU_CONTEXT_IMAGE_1= 3;  
private final static int MENU_CONTEXT_IMAGE_2= 4;  
  
@Override  
public void onCreateContextMenu(ContextMenu menu, View v,  
        ContextMenuInfo menuInfo) {  
    // Nous vérifions l'identifiant de la vue envoyée.  
    // Si celle-ci correspond à celle que nous souhaitons,  
    // nous créons nos menus.  
    // Si vous avez plusieurs vues à faire réagir, il vous suffira  
    // d'ajouter les conditions et le code nécessaire  
    // pour afficher ces même menus ou d'autre menus.  
    switch (v.getId()) {  
        case R.id.monTexteContextMenu:  
            {  
                menu.setHeaderTitle("Menu contextuel");  
                menu.setHeaderIcon(R.drawable.icon);  
                menu.add(0, MENU_CONTEXT_IMAGE_1, 0, "Mon menu contextuel 1");  
                menu.add(0, MENU_CONTEXT_IMAGE_2, 0, "Mon menu contextuel 2");  
                break;  
            }  
        case R.id.monImageContext:  
            {  
                menu.setHeaderTitle("Menu contextuel Image");  
                menu.setHeaderIcon(R.drawable.icon);  
                menu.add(0, MENU_CONTEXT_IMAGE_1, 0, "Mon menu contextuel IMAGE 1");  
                menu.add(0, MENU_CONTEXT_IMAGE_2, 0, "Mon menu contextuel IMAGE 2");  
                break;  
            }  
    }  
    super.onCreateContextMenu(menu, v, menuInfo);  
}  
  
@Override  
public boolean onContextItemSelected(MenuItem item) {  
    // Grâce à l'identifiant de l'élément  
    // sélectionné dans notre menu nous  
    // effectuons une action adaptée à ce choix  
    switch (item.getItemId()) {  
        case MENU_CONTEXT_IMAGE_1:  
            {  
                // Action pour l'option 1  
            }  
        case MENU_CONTEXT_IMAGE_2:  
            {  
                // Action pour l'option 2  
            }  
    }  
    return true;  
}
```

```
        Toast.makeText(Main.this, "Menu contextuel 1 cliqué !",
                    Toast.LENGTH_SHORT).show();
    break;
}
case MENU_CONTEXT_2:
{
    Toast.makeText(Main.this, "Menu contextuel 2 cliqué !",
                Toast.LENGTH_SHORT).show();
    break;
}
case MENU_CONTEXT_IMAGE_1:
{
    Toast.makeText(Main.this, "Menu contextuel IMAGE 1 cliqué !",
                Toast.LENGTH_SHORT).show();
    break;
}
case MENU_CONTEXT_IMAGE_2:
{
    Toast.makeText(Main.this, "Menu contextuel IMAGE 2 cliqué !",
                Toast.LENGTH_SHORT).show();
    break;
}
}
return super.onContextItemSelected(item);
}
...

```



Créer un menu contextuel

- Désormais, en cliquant quelques secondes sur l'image, un menu contextuel dédié s'affichera et les clics de l'utilisateur sur les éléments de ce menu seront gérés.



Notifier l'utilisateur par le mécanisme des « toasts »



toasts

- Notifier un utilisateur ne signifie pas le déranger alors qu'il utilise une autre application que la vôtre ou qu'il rédige un SMS. À cette fin,
- la plate-forme Android propose le concept de toast, ou message non modal s'affichant **quelques secondes** tout au plus.
- De cette façon, ni l'utilisateur ni l'application **active** à ce moment ne sont **interrompus**.



toasts

- L'utilisation d'un toast est rendue aisée par les méthodes statiques de **la classe Toast** et notamment **makeText**.
- Il vous suffit de passer **le Context** de l'application, **le texte** et **la durée d'affichage** pour créer une instance de Toast que vous pourrez afficher avec la méthode **show** autant de fois que nécessaire.



toasts

Code 11-17 : Crédit d'un toast

```
// La chaîne représentant le message  
String message = "Vous prendrez bien un toast ou deux ?";  
// La durée d'affichage (LENGTH_SHORT ou LENGTH_LONG)  
int duree= Toast.LENGTH_LONG;
```

```
// Crédit du Toast (le contexte est celui de l'activité ou du service)  
Toast toast = Toast.makeText(this, message, duree);  
// Affichage du Toast  
toast.show();
```

La durée du toast ne peut être librement fixée : elle peut soit prendre la valeur LENGTH_SHORT (2 secondes) ou LENGTH_LONG (5 secondes).

L'extrait de code précédent affiche le résultat suivant :

toasts

Figure 11–2
Affichage d'un toast s'exécutant depuis un service : l'utilisateur est notifié sans entraver son utilisation.





Positionner un toast

- Par défaut, Android affiche le message en **bas de l'écran** de l'utilisateur.
- Ce comportement peut être redéfini pour en changer la position.
- Vous pouvez spécifier la disposition d'un toast en spécifiant son ancrage sur l'écran ainsi qu'un décalage sur l'axe horizontal et vertical.



Positionner un toast

Code 11-18 : Positionner un toast

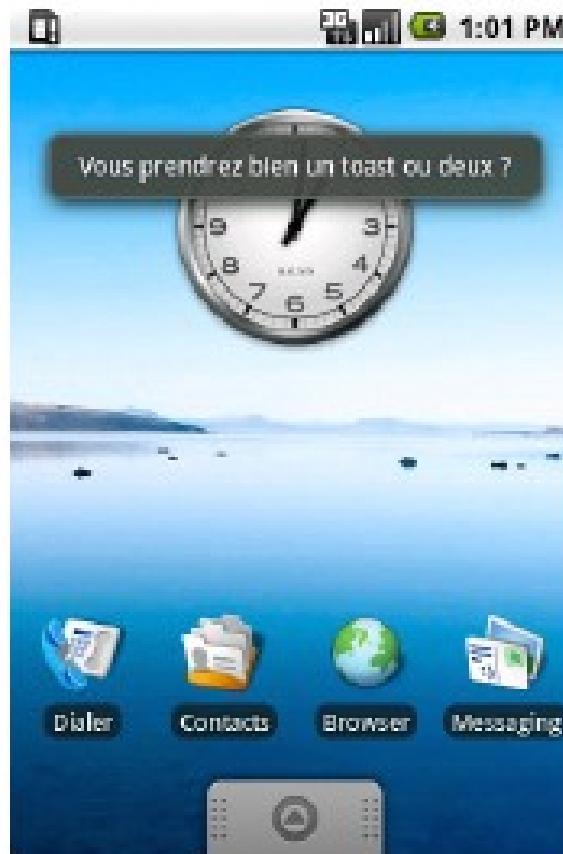
```
// Création du Toast
Toast toast = Toast.makeText(this, "Vous prendrez bien un toast ou deux ;",
Toast.LENGTH_LONG);
// Spécifie la disposition du Toast sur l'écran
toast.setGravity(Gravity.TOP, 0, 40);
// Affichage du Toast
toast.show();
```

- Le code précédent affiche le message en haut de l'écran avec un décalage vertical.



Figure 11–3

Modifiez la gravité du toast pour le positionner à un endroit approprié.



- La position du toast est bien sûr fonction de l'importance du message.
- Affiché plus haut sur l'écran, le message aura plus de chance d'attirer l'attention de l'utilisateur que s'il se trouve tout en bas.



Personnaliser l'apparence d'un toast

- Par défaut, Android utilise **une fenêtre grisée** pour afficher le texte du toast à l'utilisateur.
- Mais ce comportement n'est pas toujours le plus adapté pour communiquer des informations complexes à l'utilisateur.
- Dans certains cas, un affichage **graphique** sera plus approprié qu'un affichage **textuel**.



Personnaliser l'apparence d'un toast

- Sachez qu'Android est capable d'utiliser une vue personnalisée conçue pour l'occasion, en lieu et place du « **carré grisâtre aux bords arrondis** » par défaut.
- Pour spécifier cette vue à afficher, utilisez la méthode **setView** de l'**objet Toast**.
- Même si vous ne souhaitez pas afficher autre chose que du texte, le simple fait de personnaliser le design de la vue vous permettra de **distinguer clairement l'origine des messages**.

Intents

Utilisation, Exemples...

Définition et Utilisations

Intents

- Une application Android peut contenir plusieurs activités :
 - Une activité utilise la méthode *setContentView* pour s'associer avec une interface graphique
 - Les activités sont indépendantes les unes des autres, cependant, elles peuvent collaborer pour échanger des données et des actions
 - Typiquement, l'une des activités est désignée comme étant la première à être présentée à l'utilisateur quand l'application est lancée : on l'appelle l'activité *de démarrage*
 - Les activités interagissent en mode *asynchrone*.
 - Le passage d'une activité à une autre est réalisé en demandant à l'activité en cours d'exécuter un *Intent*.

Définition et Utilisations

Intent

- Un intent est un message qui peut être utilisé pour demander une action à partir d'un autre composant de l'application
- Un Intent permet invoquer des Activités, des *Broadcast Receivers* ou des *Services*. Les différentes méthodes utilisées pour appeler ces composantes sont les suivantes :
 - `startActivity(intent)` : lance une activité
 - `sendBroadcast(intent)` : envoie un *intent* à tous les composants *Broadcast Receivers* intéressés
 - `startService(intent)` ou `bindService(intent, ...)` : communiquent avec un service en arrière plan.

Construction d'un Intent

Intent

- Un intent comporte des informations que le système Android utilise
- **Nom du composant à démarrer**
- **Action à réaliser**
 - ACTION-VIEW, ACTION_SEND...
- **Donnée**
 - URI référençant la donnée sur laquelle l'action va agir
- **Catégorie**
 - Information supplémentaire sur le type de composants qui va gérer l'intent
 - CATEGORY-BROWSABLE, CATEGORY-LAUNCHER...
- **Extras**
 - Paires clef-valeur qui comportent des informations additionnelles pour réaliser l'action demandée
- **Drapeaux (Flags)**
 - Définissent la classe qui fonctionne comme métadonnée pour cet intent
 - Peuvent indiquer comment lancer une activité, comment la traiter une fois lancée

Types d'Intent

Intent

- Il existe deux types d'Intents
 - Intents Explicites:
 - Spécifient le composant à démarrer par nom (nom complet de la classe)
 - Permettent de démarrer un composant de votre propre application, car le nom de la classe est connu
 - *Exemple:* démarrer une activité en réponse à l'action d'un utilisateur
 - Intents Implicites
 - Ne nomment pas un composant spécifique, mais déclarent une action à réaliser
 - Permet à un composant d'une application d'appeler un composant d'une autre application
 - *Exemple:* montrer à l'utilisateur un emplacement sur une Map

Intents

INTENTS IMPLICITES

Arguments et Utilisation

Intents Implicites

- Les principaux arguments d'un Intent implicite sont :
 - Action : l'action à réaliser, peut être prédéfinie (*ACTION_VIEW*, *ACTION_EDIT*, *ACTION_MAIN*...) ou créée par l'utilisateur.
 - Donnée : Les données principales sur lesquelles on va agir, tel que le numéro de téléphone à appeler.
- Il est typiquement appelé comme suit:

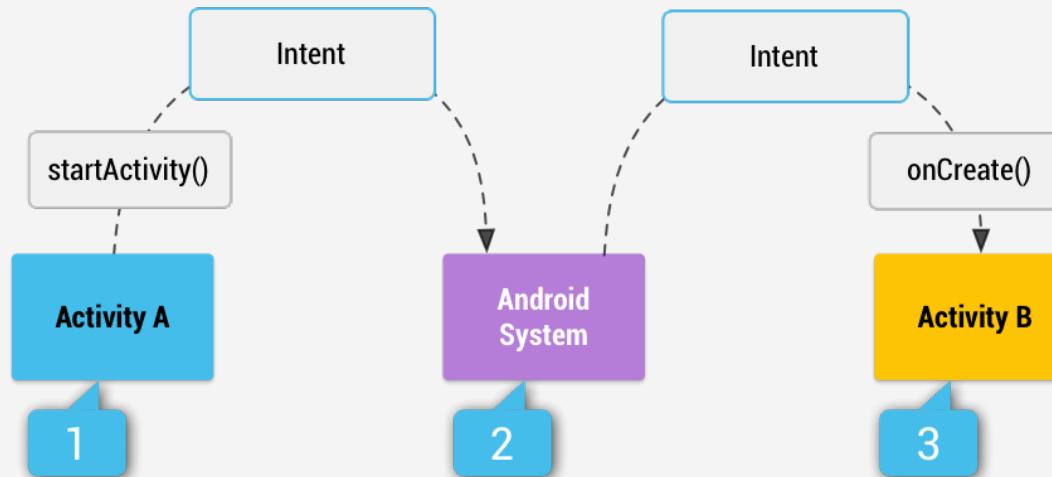
```
Intent myActivityIntent = new Intent (<action>, <donnee>) ;  
startActivity (myActivityIntent) ;
```



Comportement d'un Intent Implicit

Intents Implicites

- Un intent implicite se comporte comme suit:
 1. Activité A crée un Intent avec une action et le passe en paramètre à `startActivity`
 2. Le système Android cherchent toutes les applications pour trouver un *Intent Filter* qui correspond à cet Intent
 3. Quand une correspondance est trouvée, le système démarrent l'activité (Activity B) en invoquant sa méthode `onCreate` et en lui passant l'intent



IntentFilters

Intents Implicites

- Un Intent Filter est une expression dans le fichier Manifest d'une application qui spécifie le type d'intents que le composant veut recevoir
- Permet aux autres activités de lancer directement votre activité en utilisant un certain Intent
- Si vous ne déclarez pas d'Intent Filters à votre activité, elle ne pourra être déclenchée que par un Intent Explicite
- Il est recommandé de ne pas déclarer d'Intent Filters pour les services, car cela peut causer des problèmes de sécurité

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

Actions et Données Prédéfinies d'un Intent

Intents Implicites

Voici des exemples d'actions prédéfinies communément utilisées

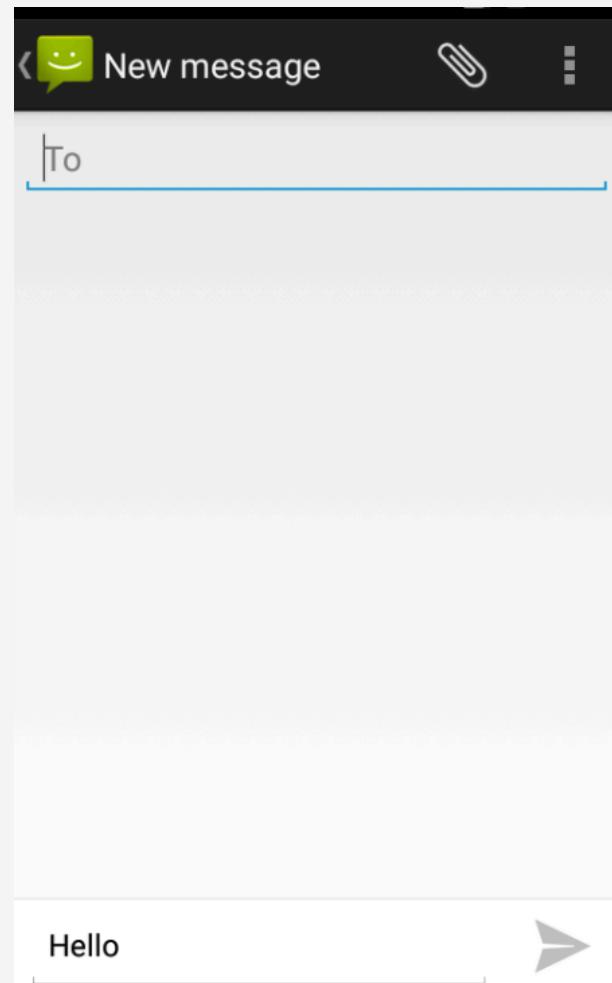
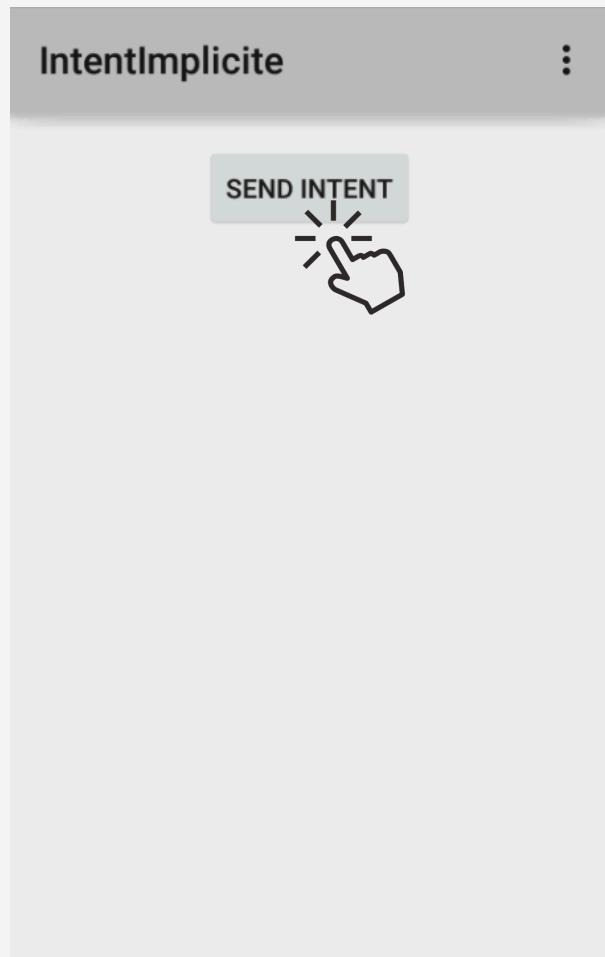
Action	Donnée	Description
ACTION_DIAL	tel:123	Affiche le numéroteur téléphonique avec le numéro (123) rempli
ACTION_VIEW	http://www.google.com	Affiche la page Google dans un navigateur.
ACTION_EDIT	content://contacts/people/2	Edite les informations sur la personne dont l'identifiant est 2 (de votre carnet d'adresse)
ACTION_VIEW	content://contacts/people/2	Utilisé pour démarrer une activité qui affiche les données du contact numéro 2
ACTION_VIEW	content://contacts/people	Affiche la liste des contacts, que l'utilisateur peut parcourir. La sélection d'un contact permet de visualiser ses détails dans un nouvel Intent.

Intent Implicit

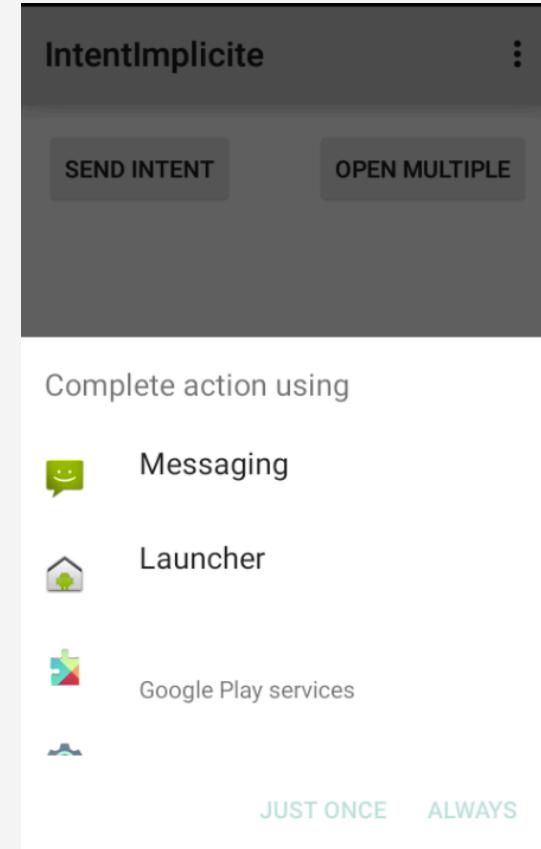
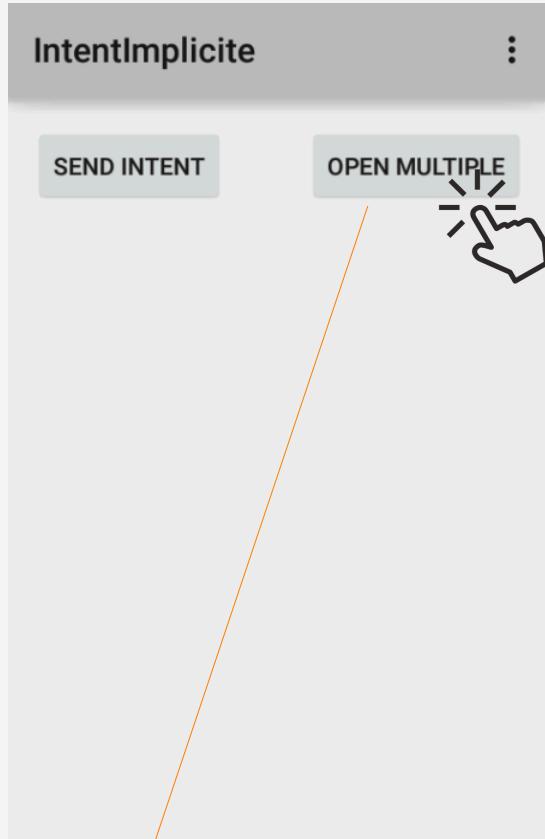
```
public void send(View v){  
    // Create the text message with a string  
    Intent sendIntent = new Intent();  
    sendIntent.setAction(Intent.ACTION_SEND);  
    sendIntent.putExtra(Intent.EXTRA_TEXT, "Hello");  
    sendIntent.setType("text/plain");  
  
    // Verify that the intent will resolve to an activity  
    if (sendIntent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(sendIntent);  
    }else{  
        Toast.makeText(this,"The send action could not be performed!",Toast.LENGTH_SHORT).show();  
    }  
}
```

Eviter que l'application crash si l'activité appelée n'existe pas

Intent Implicit



Intent Implicite: Plusieurs Activités Possibles



```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_MAIN);
```

Intents

INTENTS EXPLICITES

Arguments et Utilisation

Intents Explicites

- Les principaux arguments d'un Intent explicite sont :
 - Le contexte déclenchant l'Intent (en général *this*, si on le lance à partir de l'activité de départ, ou bien *<Activity_class_name>.this*)
 - La classe destination (en général *<Activity_class_name>.class*)
- Il est typiquement appelé comme suit:

```
Intent myActivityIntent = new Intent (StartClass.this, EndClass.class) ;  
startActivity (myActivityIntent) ;
```

Intent Explicite

```
public void start(View v){  
    Intent i = new Intent(this, DestinationActivity.class);  
    i.putExtra("name", name.getText().toString());  
    startActivity(i);  
}
```

IntentExplicite

Lilia

START INTENT



```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_destination);  
  
    tv = (TextView)findViewById(R.id.show);  
  
    String name = getIntent().getStringExtra("name");  
    tv.setText("Hello "+name+"!");  
}
```

DestinationActivity

Hello Lilia!

Démarrer une Activité avec Résultat

Intents Explicites

- Il est possible d'établir un lien bidirectionnel entre deux activités grâce à un Intent
- Pour recevoir un résultat à partir d'une autre activité, appeler *startActivityForResult* au lieu de *StartActivity*
- L'activité destination doit bien sûr être conçue pour renvoyer un résultat une fois l'opération réalisée
- Le résultat est envoyé sous forme d'Intent
- L'activité principale le recevra dans un *onActivityResult*

Intent avec Résultat

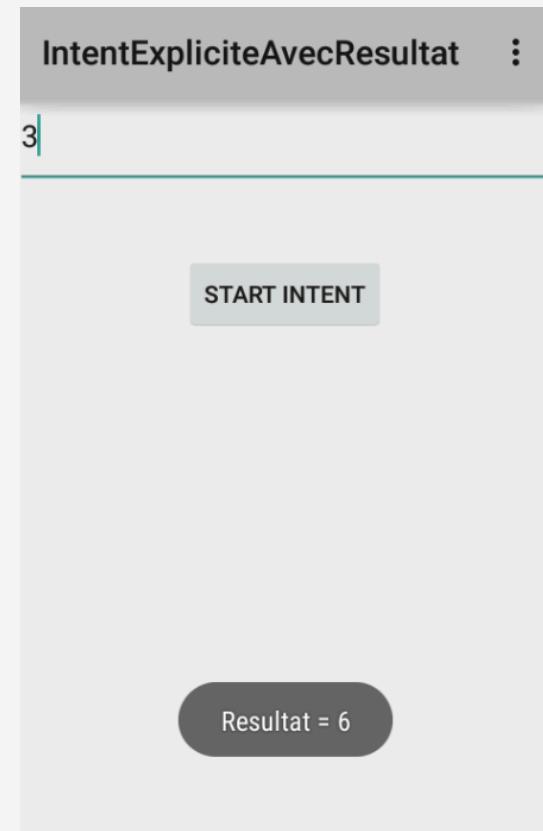
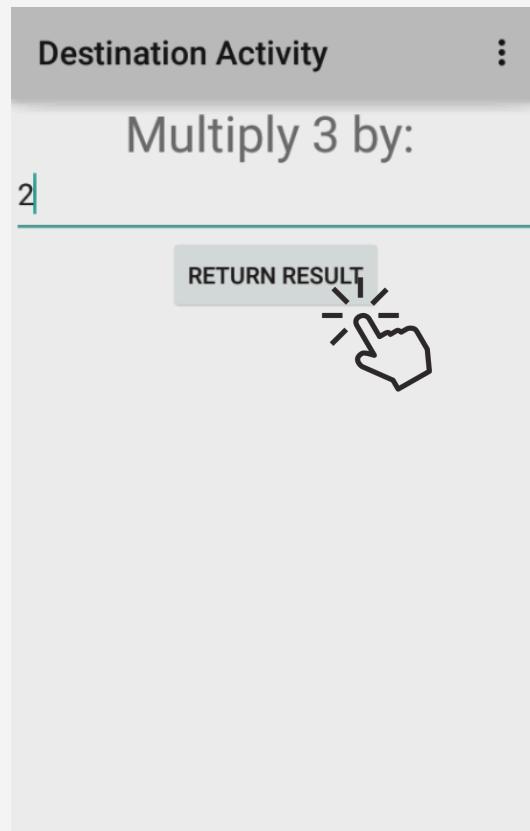
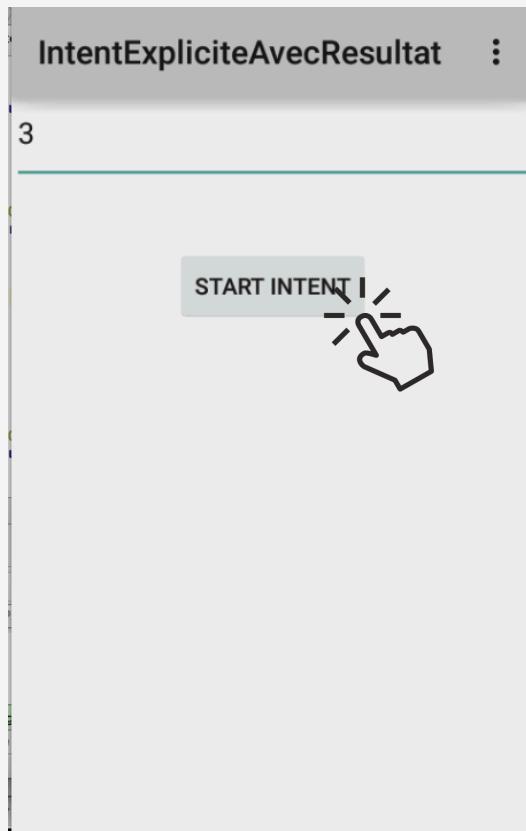
```
public void start(View v){  
    Intent i = new Intent(this, DestinationActivity.class);  
    i.putExtra("val", Integer.valueOf(texte.getText().toString()));  
    startActivityForResult(i, REQUEST_CODE);  
}  
  
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {  
  
    if (requestCode == REQUEST_CODE) {  
        if(resultCode == RESULT_OK){  
            Toast.makeText(this,"Résultat = "+intent.getIntExtra("resultat",0),Toast.LENGTH_LONG).show();  
        }  
        if (resultCode == RESULT_CANCELED) {  
            Toast.makeText(this,"Pas de Résultat !",Toast.LENGTH_LONG).show();  
        }  
    }  
}
```

Activité 1

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_destination);  
  
    tv = (TextView)findViewById(R.id.show);  
    mult = (EditText) findViewById(R.id.multEdit);  
  
    valueReceived = getIntent().getIntExtra("val",0);  
    tv.setText("Multiply "+valueReceived+" by: ");  
}  
  
public void retour (View v){  
    Intent returnIntent = new Intent();  
    if (mult.getText()!= null) {  
        int result = valueReceived * Integer.valueOf(mult.getText().toString());  
        returnIntent.putExtra("resultat", result);  
        setResult(RESULT_OK, returnIntent);  
    }else{  
        setResult(RESULT_CANCELED, returnIntent);  
    }  
    finish();  
}
```

Activité 2

Intent avec Résultat



Chapitre 7

Persistance et partage des données

1 Introduction

Android offre plusieurs méthodes pour stocker les données d'une application.

La solution choisie va dépendre des besoins : données privées ou publiques, un petit ensemble de données à préserver ou un large ensemble à préserver localement ou via le réseau.

Ces méthodes utilisent soit des éléments propres à l'API Java ou ceux associés à l'API d'Android.

Ces méthodes sont :

Persistance dans l'état d'une application

On utilise pour cela la notion de « Bundle » et les différents cycles de l'activité pour sauvegarder l'information utile à l'aide du « bundle » et récupérer cette information dans un autre état de l'activité.

On ne peut utiliser qu'un seul bundle, par ailleurs la donnée n'est pas persistante et n'est disponible que tant que l'application est utilisée.

Préférences partagées

Un ensemble de paires : clé et valeur. Clé est un « String » et Valeur est un type primitif (« boolean », « String », etc.).

Ces préférences sont gérées à travers un code Java ou bien via une activité.

Les données ne sont pas cryptées.

Les préférences sont adaptées pour des paires simples, mais dès qu'il est question de données plus complexes, il est préférable d'utiliser des fichiers.

Fichiers (création et sauvegarde)

Android permet la création, la sauvegarde et la lecture de fichiers à travers un média persistant (mémorisation et disponibilité).

Les fichiers peuvent être de n'importe quel type (image, XML, etc.).

Les fichiers peuvent être considérés pour une utilisation interne, donc local à l'application, ou bien externe, donc partagée avec plusieurs applications.

Base de données relationnelle, SQLite

Android offre aussi la possibilité d'utiliser toutes les propriétés d'une base de données relationnelle.

Android utilise pour cela une base de données basée sur « SQLite » (www.sqlite.org).

Android stocke la base de données localement à l'application.

Si l'on veut partager cette structure de données avec d'autres applications, il faudra utiliser dans ce cas, un gestionnaire de contenu (content provider) configuré à cet effet.

Stockage réseau

Android permet de stocker des fichiers sur un serveur distant.

On peut utiliser pour cela les différentes techniques examinées dans le chapitre « Internet ».

2 Persistance dans l'état d'une application

Android peut arrêter l'activité et la redémarrer quand il y a :

- Rotation de l'écran.
- Changement de langue.
- L'application est en arrière-plan et le système a besoin de ressources.
- Et quand vous cliquez le bouton « retour » (« back »).

Ce redémarrage peut provoquer la perte des changements apportés à votre activité.

Une solution consiste à préserver les données dans un bundle à travers la méthode `onSaveInstanceState` puis récupérer cette information par la suite à l'aide de la méthode `onRestoreInstanceState` (ou bien dans la méthode `onCreate`).

Cette solution n'est pas appropriée dans le cas d'un clic sur le bouton « retour ». Dans ce cas, l'application démarre à partir de zéro et les données préservées sont détruites.

Généralement, l'orientation est gérée par la création d'une vue appropriée. Mais supposons que vous n'ayez pas eu le temps de le faire!

Nous pouvons fixer dans le fichier « `AndroidManifest.xml` » l'orientation supportée, comme suit :

```
<activity android:name=".YourActivity"  
        android:label="@string/app_name"  
        android:screenOrientation="portrait">
```

On déclare un widget « EditText », on insère une valeur quelconque, puis on change l'orientation de l'écran (CTRL-F11/CTRL-F12) :

```
<EditText android:layout_width="fill_parent"  
        android:layout_height="wrap_content"/>
```



On constate que l'on a perdu le contenu du Widget en changeant l'orientation de l'écran.

Si on avait ajouté dans le fichier manifeste cette ligne « `android:configChanges="orientation|screenSize"` », le contenu du Widget va être préservé. Cependant si une vue paysage a été définie avec l'application, Android ne va pas la prendre en considération.

On reprend la même activité et on identifie maintenant le Widget, « EditText », avec un identificateur comme suit :

```
<EditText android:id="@+id/edittext1"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"/>
```

On teste de nouveau l'activité et on remarque maintenant que le contenu du Widget a été préservé.



Le fait, de passer d'une orientation à une autre, a fait en sorte que l'activité a été arrêtée puis démarrée de nouveau. Sauf que cette fois-ci, le contenu du Widget n'a pas été remis à zéro.

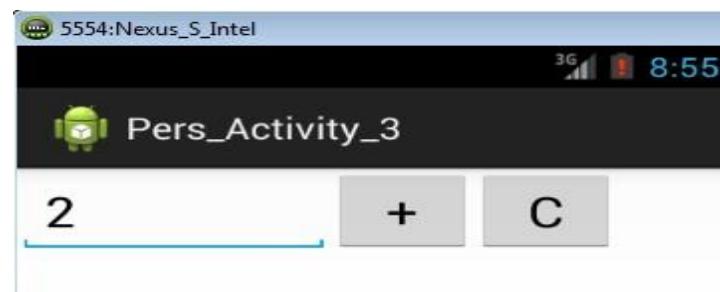
Android préserve automatiquement l'état d'un Widget quand ce dernier est identifié. Ce qui est le cas dans le second exemple.

Prenons un autre exemple, une simple calculatrice.

Faites $2 + 2 +$, le résultat sera 4. Faites tourner l'orientation de l'écran, le résultat va rester 4. À noter que le bouton « C » permet de remettre à zéro la calculatrice.



Faites maintenant le test suivant : appuyez sur les touches $2 + 2$, puis changez l'orientation de l'écran.



Appuyez maintenant sur la touche + et examinez le résultat :



Le résultat final est égal à 2 et non pas 4! Un bogue!

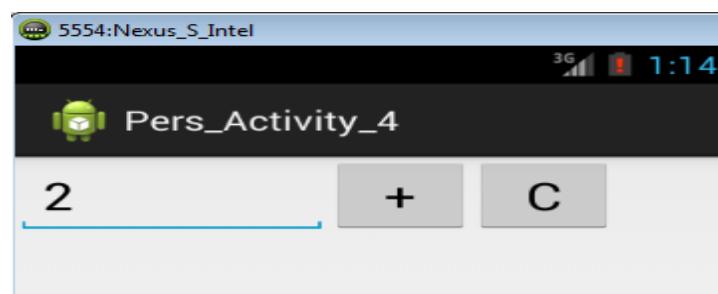
En changeant d'orientation, l'activité a été arrêtée puis démarrée de nouveau. Durant cette opération, nous avons perdu la valeur de la somme intermédiaire, ce qui explique pourquoi la somme finale affichée est erronée.

Dans le code Java associé à cette activité, la somme intermédiaire est représentée par la variable « total ». Il faudra donc préserver l'état de cette variable au moment de changer d'orientation.

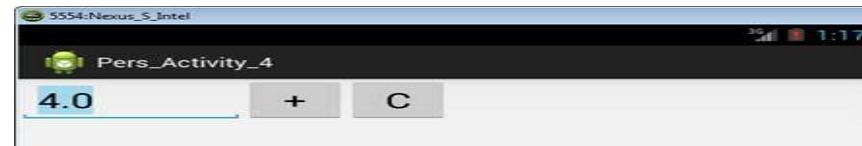
Pour ce faire, nous allons utiliser la méthode « `onSaveInstanceState` » pour préserver la valeur associée à cette variable au moment de l'arrêt de l'application.

On peut récupérer cette valeur lors du démarrage de l'activité à l'aide de la méthode « `onRestoreInstanceState` ».

```
@Override  
public void onSaveInstanceState(Bundle outState) {  
    super.onSaveInstanceState(outState);  
    outState.putFloat("TOTAL", total);  
}  
  
@Override  
public void onRestoreInstanceState(Bundle savedInstanceState) {  
    super.onRestoreInstanceState(savedInstanceState);  
    total=savedInstanceState.getFloat("TOTAL");  
}
```



On clique sur le signe + :



Le total est maintenant correct.

3 Préférences partagées

Cette méthode permet de sauvegarder les préférences dans un fichier.

Ces préférences seront accessibles aux différentes activités associées à l’application.

Les préférences seront utilisées pour sauvegarder l’état de la configuration de l’application ou bien les données de session.

Les préférences sont comme les « bundles » sauf qu’elles sont persistantes ce qui n’est pas le cas des bundles.

Les préférences ne sont pas cryptées. Il faudra faire attention si l’intention est de préserver des données critiques.

Les préférences peuvent être effacées par l’utilisateur de l’application.

Chaque préférence a la forme d'une paire dont la clé est un élément du type String et dont la valeur est un des types primitifs (int, long, float, boolean) ou bien une collection de String (Set<String>).

Cette méthode n'est appropriée que pour une petite collection de paires.

Nous allons utiliser les méthodes de la classe « SharedPreferences » pour sauvegarder puis lire une préférence.

Une instance de la classe « SharedPreferences » est créée dans un mode prédéfini. Le mode le plus couramment utilisé est « MODE_PRIVATE » pour signifier que les préférences ne seront accessibles que par l'application.

Il existe d'autres modes :

- MODE_WORLD_READABLE : les autres applications peuvent lire l'ensemble,
 - MODE_WORLD_WRITEABLE : les autres applications peuvent modifier l'ensemble,
 - MODE_MULTI_PROCESS : plusieurs processus peuvent accéder à l'ensemble.
-

Les modes « WORLD_READABLE » et « WORLD_WRITEABLE » sont dépréciés depuis l’API 17. Ils provoqueront l’exception « SecurityException » depuis l’API 24 (Nougat).

Création :

Nous pouvons utiliser l’une des deux méthodes :

getPreference(int mode) : elle est associée à l’activité courante. Une procédure transparente est définie par défaut pour préserver les préférences désirées et associées à l’activité. Nous devons juste signifier l’argument qui représente l’un des modes d’accès précédemment mentionnés.

```
private SharedPreferences settings =  
    getPreference(context.MODE_PRIVATE);
```

getSharedPreferences(String nom,int mode) : elle est utilisée si les préférences sont préservées dans plusieurs fichiers. Le premier argument représente le nom du fichier à utiliser et le second argument, le mode d'accès.

```
private SharedPreferences settings =  
    getSharedPreferences("nom_preferences", context.MODE_PRIVATE);
```

On édite la préférence:

```
SharedPreferences.Editor editor = settings.edit();
```

On définit la paire « string,string » et on valide l'inscription dans le conteneur des préférences :

```
editor.putString(NOM, PrefValeur);  
editor.apply();
```

Pour récupérer la paire :

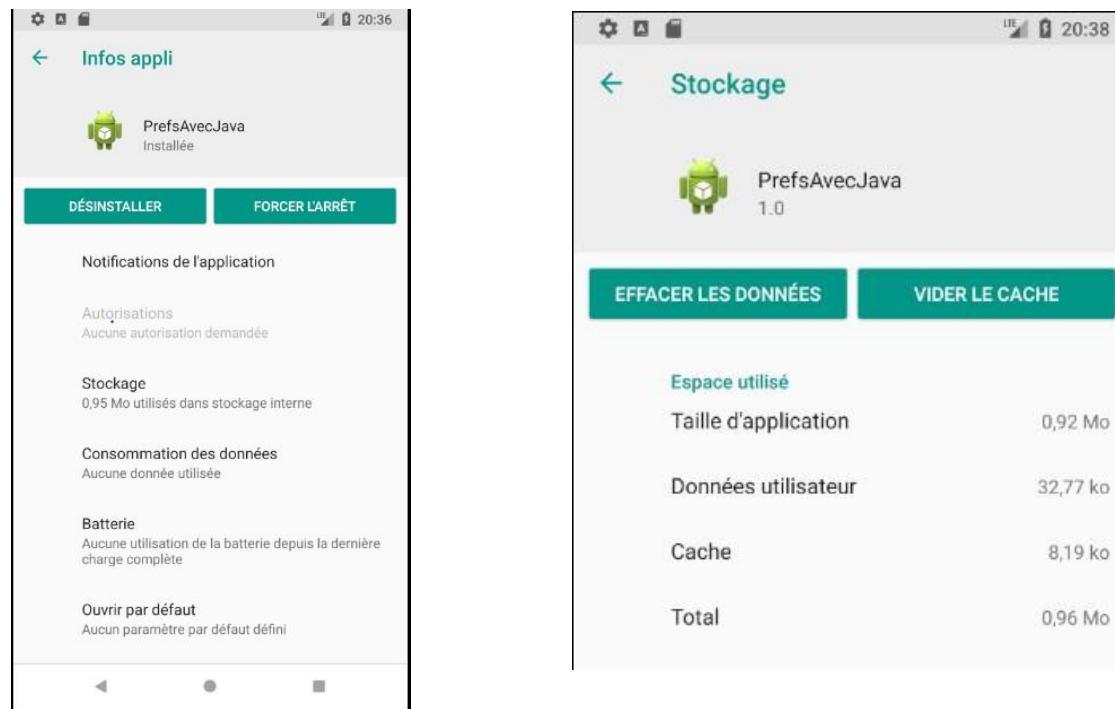
```
PrefValeur = parametres.getString(NOM, "Introuvable");
```

On peut détruire la préférence en détruisant la clé associée à l'aide de la méthode « removeString(String clé) ». On peut détruire toutes les préférences à l'aide de la méthode « clear » :

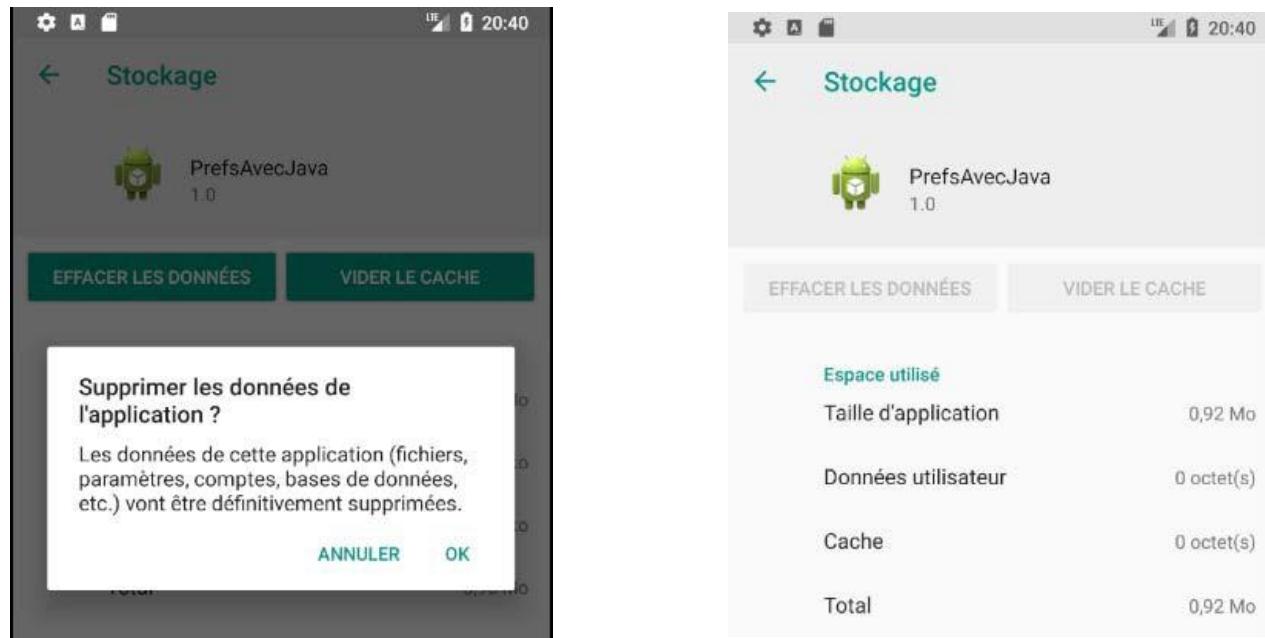
```
editor.removeString(Nom);  
editor.clear();
```

L'utilisateur peut aussi détruire les préférences sauvegardées par son application en procédant ainsi sous « Pie » :

- On se positionne sur le menu des applications.
 - On sélectionne notre application et on maintient le bouton gauche de la souris enfoncé. Une nouvelle fenêtre va apparaître. Cliquer sur « App info ». Vous allez obtenir ce qui suit :
-



Vous cliquer par la suite sur « Stockage ». Vous pouvez effacer les préférences en cliquant sur le bouton « EFFACER LES DONNÉES ».



Une autre manière de procéder est de passer par « settings », puis « Apps », puis sélectionnez votre application « PrefsAvecJava ».

Les préférences sont sauvegardées dans un fichier. Ce fichier a par défaut le format « XML » et est localisé à :

/data/data/cis493.preferences/shared_prefs/MyPreferences_001.xml

Sur l’émulateur, vous pouvez voir l’existence du fichier grâce à la vue « Device File Explorer ». Vous pouvez aussi extraire le fichier à l’aide de « adb » et examiner son contenu. Il faut avoir les privilèges d’un utilisateur « root » pour pouvoir y avoir accès. Pour obtenir ces privilèges dans le cadre d’un émulateur, exécutez la commande :

```
adb root
```

```
restarting adbd as root
```

```
adb pull
```

```
/data/data/cis493.preferences/shared_prefs/MyPreferences_001.xml .
```

Choisissez en premier « Pref Simple UI » et examinez par la suite le contenu du fichier « MyPreferences_001.xml » :

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<int name="textSize" value="15" />
<int name="backColor" value="-1" />
</map>
```

Choisissez par la suite « Pref Fancy UI » et examinez le contenu du fichier « MyPreferences_001.xml » :

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<int name="layoutColor" value="-16711936" />
<int name="textSize" value="20" />
<int name="backColor" value="-16776961" />
<string name="textStyle">bold</string>
</map>
```

Cliquez maintenant sur le bouton retour (« Back ») puis examinez le contenu du fichier « MyPreferences_001.xml » :

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <int name="layoutColor" value="-16711936" />
    <string name="DateLastExecution">3 févr. 2018</string>
    <int name="textSize" value="20" />
    <int name="backColor" value="-16776961" />
    <string name="textStyle">bold</string>
</map>
```

Sauvegarde des préférences à travers une activité

L'exemple crée une interface pour gérer les préférences de l'utilisateur.

En utilisant ce programme avec, par exemple, l'API 17, nous obtenons cet avertissement pour l'appel :

```
addPreferencesFromResource(R.xml.prefs);
```

[The method addPreferencesFromResource(int) from the type PreferenceActivity is deprecated].

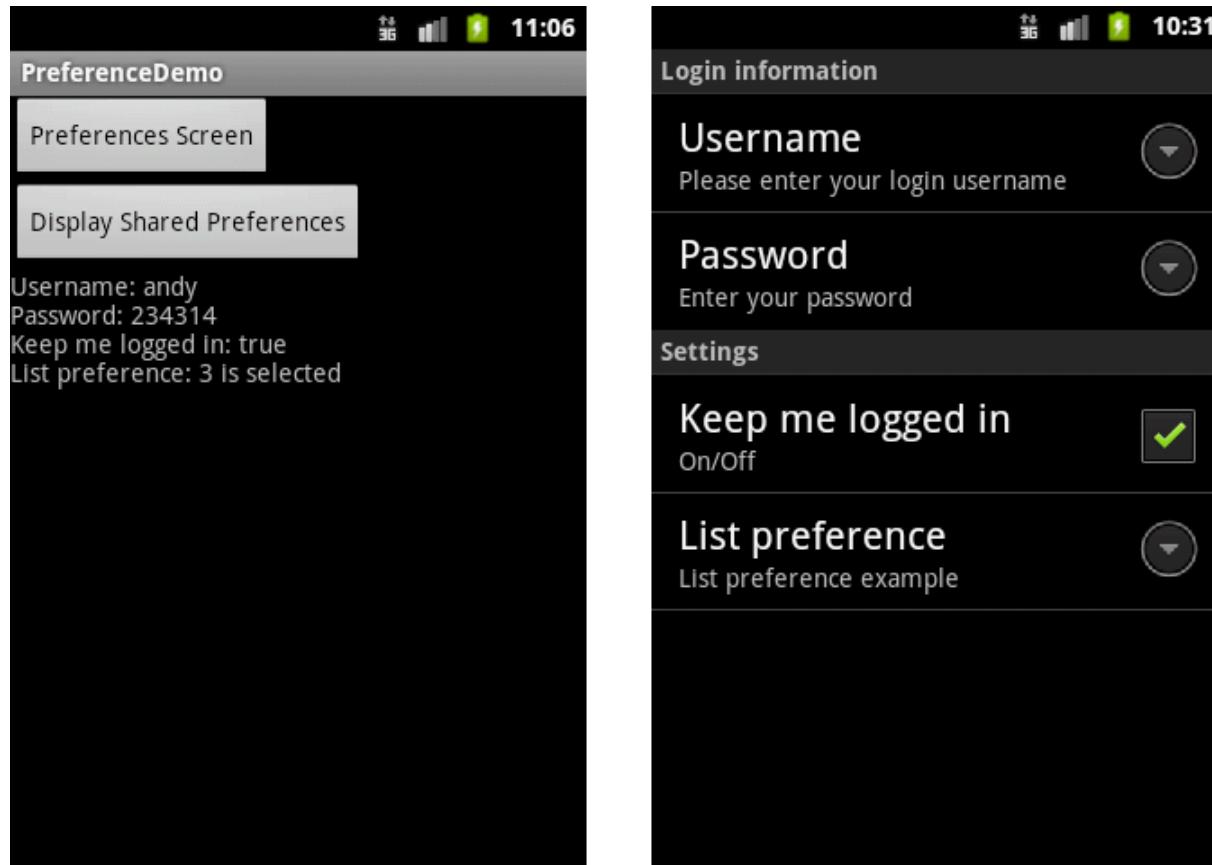
La méthode « `addPreferencesFromResource` » de la classe « `PreferenceActivity` » a été déclarée désuète à partir de l’API 11 (Honeycomb).

Nous devons utiliser à la place la méthode équivalente définie dans la classe « `PreferenceFragment` ».

Nous avons dû donc apporter des modifications à cet exemple pour qu'il soit compatible, peu importe la version de l’API utilisé.

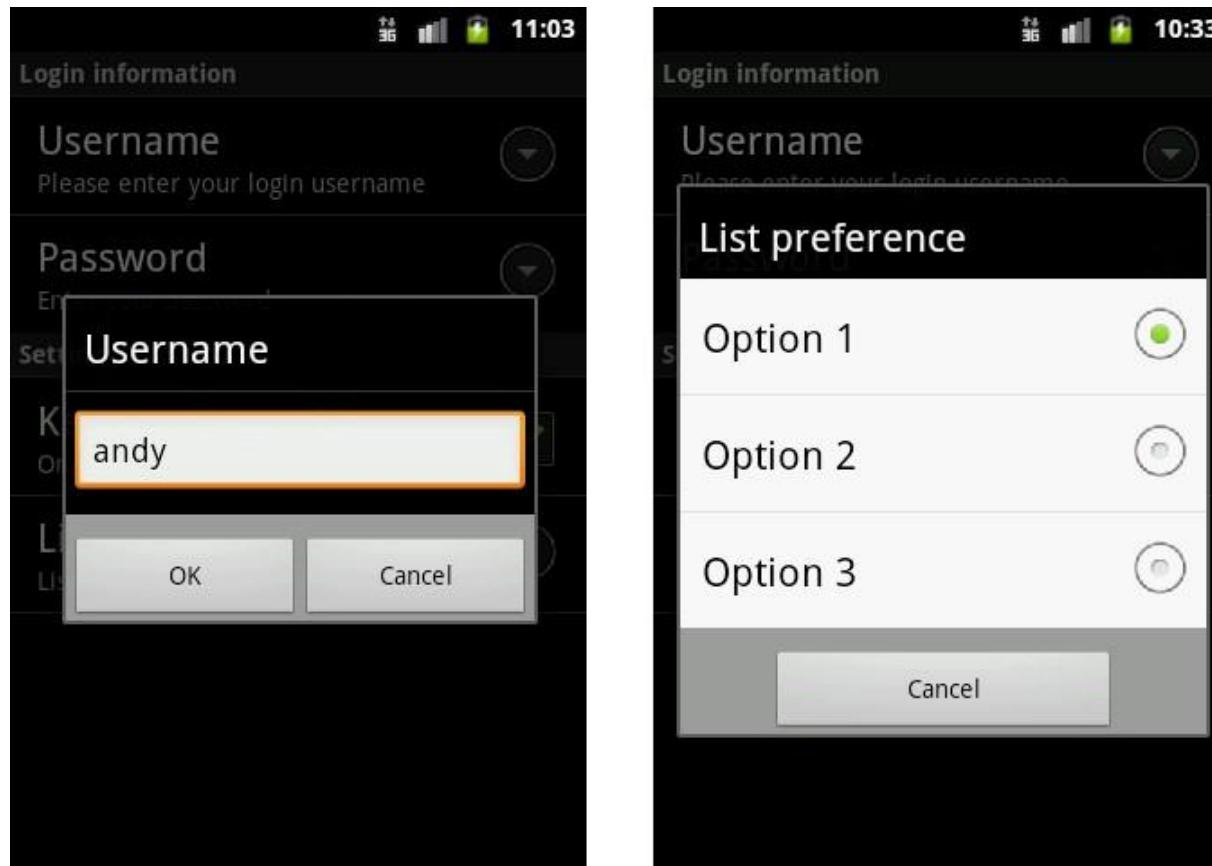
Un test d’API va permettre de choisir entre « `PreferenceActivity` » et « `PreferenceFragment` ».

Nous allons commencer par décrire brièvement l'exemple :



En cliquant sur le bouton « Preferences Screen », une nouvelle activité va être lancée (écran de droite).

Quand l'utilisateur choisit une des options ...



Écran de gauche, nom d'utilisateur; écran de droite, la liste des préférences.

Le contenu de cette 2^e activité se trouve dans le fichier « prefs.xml », dans le répertoire « res/XML ». Nous avons ajouté aussi le fichier « PrefsActivity.java », qui contient cet appel dans la méthode « onCreate » :

```
addPreferencesFromResource(R.xml.prefs);
```

Cette méthode permet de télécharger des préférences à partir d'une ressource, décrite dans un fichier « XML ». Ce fichier va définir la vue de l'activité.

Examiner le répertoire « xml » dans la hiérarchie du projet.

Dans le fichier « prefs.xml », le tag « PreferenceScreen » sera utilisé comme la racine du fichier. Quand une activité pointe ce fichier, le tag « PreferenceScreen » sera utilisé comme le point d'entrée.

Nous distinguons plusieurs tags :

<PreferenceCategory>	définit une catégorie de préférences. Pour l'exemple, nous avons défini deux catégories « Login Information » et « Settings ».
<EditTextPreference>	définit un champ pour stocker de l'information.
<CheckBoxPreference>	définit une boîte à cocher, « checkbox ».
<ListPreference>	définit une liste d'éléments. La liste apparaît comme des boutons radio.

Par la suite, l'activité principale récupère automatiquement les préférences préservées :

```
SharedPreferences prefs =  
    PreferenceManager.getDefaultSharedPreferences(  
        PreferenceDemoActivity.this);
```

Pour l'API 17, nous avons développé une version avec fragments.

Nous avons défini une classe pour lire le fichier « XML » :

```
public class PrefsActivity02 extends PreferenceFragment {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        addPreferencesFromResource(R.xml.prefs);  
    }  
}
```

Nous avons apporté aussi les modifications suivantes à la classe « PrefsActivity » :

```
if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {  
    addPreferencesFromResource(R.xml.prefs);  
}else{  
    getFragmentManager().beginTransaction().replace(android.R.id.content,  
        new PrefsActivity02()).commit();  
}
```

Ainsi donc, la commande à exécuter va dépendre de la version de l'API.

4 Fichiers (création et sauvegarde)

Android s'appuie sur l'API Java pour réaliser la gestion de fichiers.

Préférences partagées

Une première utilisation des fichiers était en rapport avec les préférences partagées. Nous avons expliqué comment il était possible de paramétrier la méthode « `getSharedPreferences` » pour préserver ces préférences dans un fichier.

Stockage interne

Comme nous l'avons fait depuis le début du cours, nous pouvons ajouter des fichiers supplémentaires à l'application. Ces fichiers peuvent être disposés dans divers endroits en fonction de leur utilisation : « assets », « res/raw », etc.

Ils feront partie du paquetage « apk » final.

Pour l'exemple, nous allons d'abord préserver le fichier « my_text_file.txt » dans le répertoire « res/raw » de l'application. Par la suite, l'application va se charger de lire le fichier en question et de l'afficher dans la vue principale.

Association d'un flux à la ressource :

```
int fileResourceId = R.raw.my_text_file;  
InputStream is = this.getResources()  
                .openRawResource(fileResourceId);
```

Lecture du contenu de la ressource :

```
if (is!=null) {  
    BufferedReader reader =  
        new BufferedReader(new InputStreamReader(is));  
    while ((str = reader.readLine()) != null) {  
        buf.append(str).append("\n");  
    }  
}
```

Comme il s'agit d'une opération I/O, il faudra penser à capturer l'exception « IOException ».

Quand l'application démarre pour la première fois, nous allons d'abord écrire un texte dans le fichier « notes.txt » et cliquer sur le bouton de sauvegarde. Ce bouton va terminer l'application, mais avant cela, la méthode « onPause » sera exécutée. Cette méthode aura la tâche de sauvegarder le texte dans le fichier « notes.txt ».

Examiner le contenu du fichier « notes.txt » qui se trouve dans le répertoire :

/data/data/cis470.matos.filewriteread/files

Il faut être « root » pour y accéder au répertoire.

Quand l'activité est démarrée une seconde fois, la méthode « onStart » est exécutée. Cette méthode va se charger de lire puis d'afficher le contenu du fichier sur l'écran.

Stockage externe

Nous allons examiner comment effectuer des opérations de lecture et écriture à partir d'un média externe. Nous allons utiliser pour cela une carte mémoire SD pour effectuer ces opérations I/O.

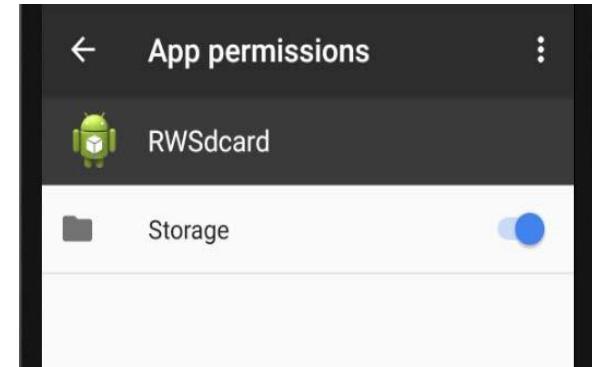
Toutes les applications ont par défaut la permission de lire à partir d'une unité externe. Ce comportement va changer dans les prochaines versions de l'API. Il est préférable donc d'ajouter la permission de lecture dès à présent, si votre application doit lire à partir d'une unité externe.

La permission d'écrire n'est pas accordée par défaut. Il faudra l'autoriser explicitement.

Ces permissions doivent être ajoutées dans le fichier « *AndroidManifest.xml* » :

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

Il faudra manuellement activer la permission de stockage de votre application :



Comme Android est basé sur un noyau Linux, il faudra vérifier que l'unité externe est bien disponible (l'expression utilisée dans le jargon est « montée »).

```
public boolean isExternalStorageWritable() {  
    String state = Environment.getExternalStorageState();  
    if (Environment.MEDIA_MOUNTED.equals(state)) {  
        return true;  
    }  
    return false;  
}
```

La méthode « `Environment.getExternalStorageState()` » retourne un « `MEDIA_MOUNTED` » pour nous informer si le média est présent, monté, et dans quel mode, lecture/écriture.

Nous testons par la suite le mode qui a été autorisé avec la méthode « `Environment.MEDIA_MOUNTED.equals` ». Si elle retourne « `true` » alors l’écriture a été autorisée. Dans le cas contraire, il faudra vérifier si la lecture a été autorisée en procédant ainsi :

```
/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

Dans notre exemple, nous avons mis en cascade les différents appels comme suit :

```
if(!Environment.MEDIA_MOUNTED.equals  
    (Environment.getExternalStorageState())){  
    Toast.makeText(this, "External SD card not mounted",  
                  Toast.LENGTH_LONG).show();  
}
```

Après avoir validé la disponibilité de l'unité externe, pour préciser le chemin à joindre au nom du fichier, nous allons faire appel à la méthode « Environment.getExternalStorageDirectory() » :

```
File root = Environment.getExternalStorageDirectory();  
File directory = new File (root.getAbsolutePath() + "/Data");  
boolean wasSuccessfulmkdirs = directory.mkdirs();  
File myFile = new File (directory, "textfile.txt");  
boolean wasSuccessfulnewfile = myFile.createNewFile();
```

Le fichier « textfile.txt » va être disponible dans le répertoire « /sdcard/Data ».

Les autres méthodes de lecture et d'écriture sont celles de l'API Java.

Nous pouvons vérifier par la suite avec la commande « adb » la présence du fichier et son contenu :

```
adb shell  
cd sdcard  
cd Data  
cat textfile.txt
```

5 SQLite

Android intègre le système de gestion de bases de données, SQLite.

Pour plus de détails, consultez ce lien : <http://www.sqlite.org/>

C'est un système compact, très efficace pour les systèmes embarqués. En effet, il utilise très peu de mémoire.

SQLite ne nécessite pas de serveur pour fonctionner, ce qui n'est pas le cas de MySQL par exemple.

Les opérations sur la base de données se feront donc dans le même processus que l'application. Il faudra faire attention aux opérations « lourdes », votre application va ressentir les contres coups. Il est conseillé dans ce cas d'utiliser les tâches asynchrones (ou threads).

Chaque application peut avoir donc ses propres bases.

Ces bases sont stockées dans le répertoire « databases » associé à l'application (/data/data/APP_NAME/databases/nom_base). Nous pouvons les stocker aussi sur une unité externe (sdcard).

Chaque base créée, elle sera en mode « MODE_PRIVATE ». Aucune autre application ne peut y accéder que l'application qui l'a créée.

Pour y avoir accès, il faut que la base ait été sauvegardée sur un support externe, sinon utiliser le mécanisme d'échange de données fourni par Android (il sera développé plus tard dans ce chapitre).

L'accès par « adb » nécessite les privilèges « root ».

SQLite supporte les types : TEXT (chaîne de caractères), INTEGER (entiers), REAL (réels). Tous les types doivent être convertis pour être utilisés. SQLite ne vérifie pas le typage des éléments. À vous de vous en assurer que vous n'avez pas écrit un entier à la place d'une chaîne de caractères par exemple.

Création et mise à jour de la base

L'exemple va créer une table de commentaires. Chaque commentaire est identifié par un identificateur unique.

Nom de la table : comments	
<u>_id</u>	comments

La base va porter le nom « **comments.db** ».

L'organisation des fichiers permet de faciliter l'organisation de la base de données et la compréhension de l'exemple.

- Le fichier « **Comment.java** » va contenir un enregistrement d'une table et les différentes méthodes qui gravitent autour.
-

La classe « Comment » décrite dans le fichier « Comment.java » contient deux attributs :

```
private long id;  
private String comment;
```

La base de données doit utiliser un identifiant unique « _id » comme clé primaire de la table. Des méthodes d'Android se servent de ce standard.

- Le fichier « MySQLiteHelper.java » contient la classe qui dérive de « SQLiteOpenHelper ».

La classe « MySQLiteHelper » :

Créez une nouvelle classe qui va dériver de la classe « SQLiteOpenHelper » :

```
public class MySQLiteHelper extends SQLiteOpenHelper { ...}
```

Dans le constructeur de la classe, faites appel à la méthode « super » de « SQLiteOpenHelper » et spécifiez le nom de la base et sa version.

```
public MySQLiteHelper(Context context) {  
    super(context, "commments.db", null, 1);  
}
```

Dans cette classe, vous devez redéfinir les méthodes « onCreate(SQLiteDatabase MaBase) » et « onUpgrade(SQLiteDatabase MaBase) ». L'argument représente votre base.

La méthode « onCreate » est appelée pour la création de la base si elle n'existe pas.

```
public void onCreate(SQLiteDatabase database) {  
    database.execSQL(DATABASE_CREATE);  
}
```

La variable « *DATABASE_CREATE* » va contenir la requête « SQL » qui permet de créer la base.

La méthode « `onUpgrade` » est appelée pour mettre à jour la version de votre base. Elle vous permet de mettre à jour le schéma de votre base.

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

Il est préférable de créer une classe par table. Cette classe va définir les méthodes « `onCreate` » et « `onUpgrade` ». Vous allégez ainsi le code de la classe qui dérive de « `SQLiteOpenHelper` ».

- Le fichier « `CommentsDataSource.java` » contient la classe contrôleur. Elle contient les différentes méthodes qui vont interagir avec la base de données. C'est le DAO (Data Access Object)

La classe « `SQLiteOpenHelper` » fournit les deux méthodes « `getReadableDatabase()` » et « `getWritableDatabase()` » pour accéder à une instance « `SQLiteDatabase` » en mode de lecture ou écriture.

Ouverture de la base :

```
public void open() throws SQLException {  
    database = dbHelper.getWritableDatabase();  
}
```

Fermeture de la base :

```
public void close() {  
    dbHelper.close();  
}
```

Insérer un élément :

Pour insérer un élément dans la base, il faut d'abord former l'enregistrement. On utilise pour cela un objet du type « ContentValues » qui représente une collection de champs.

```
public long createComment(String comment) {  
    ContentValues values = new ContentValues();  
    values.put(MySQLiteHelper.COLUMN_COMMENT, comment);  
    long insertId = database.insert(MySQLiteHelper.TABLE_COMMENTS,  
                                    null, values);  
    return insertId;  
}
```

Effacer un élément :

```
public void deleteComment(Comment comment) {  
    long id = comment.getId();  
    database.delete(MySQLiteHelper.TABLE_COMMENTS,  
                    MySQLiteHelper.COLUMN_ID + " = " + id, null);  
}
```

Faire une sélection :

```
Cursor cursor = database.query(MySQLiteHelper.TABLE_COMMENTS,  
                                allColumns, MySQLiteHelper.COLUMN_ID + " = " + insertId,  
                                null, null, null, null);
```

La méthode « query » retourne une instance de « Cursor » qui représente un ensemble de résultats.

- Le fichier « `TestDatabaseActivity.java` » contient l’activité associée à notre application.

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_testdatabase);  
  
    datasource = new CommentsDataSource(this);  
    datasource.open();  
  
}  
  
protected void onResume() {  
    super.onResume();  
    datasource.open();  
}  
  
protected void onPause() {  
    super.onPause();  
    datasource.close();  
}
```

Accès à la base de données SQLite

Le SDK d'Android inclut un programme permettant de lire une base de données SQLite.

Nous devons extraire le fichier de l'émulateur via la commande « adb/shell/pull » en tant que « root » ou bien en utilisant la vue « Device File Explorer », puis la commande « pull » :

```
adb pull  
/data/data/ca.umontreal.iro.ift1155.testdatabaseactivity/databases/comments.db
```

```
C:> sqlite3 comments.db
SQLite version 3.9.2 2015-11-02 18:31:45
Enter ".help" for usage hints.
sqlite> .tables
android_metadata  comments
sqlite> select * from comments;
1|Hate it
2|Cool
3|Very nice
4|Hate it
5|Very nice
6|Cool
sqlite> .exit
```

En tant que « root », il est possible aussi d'y avoir accès par la commande « adb » comme suit :

```
C:> adb shell
generic_x86_64:/ #
generic_x86_64:/ # cd /data/data/ca.umontreal.iro.ift1155.testdatabaseactivity
/databases/
generic_x86_64:/data/data/
ca.umontreal.iro.ift1155.testdatabaseactivity/databases
# sqlite3 comments.db
SQLite version 3.9.2 2015-11-02 18:31:45
Enter ".help" for usage hints.
Sqlite>.exit
```

On peut utiliser un add-on dans Firefox pour accéder à la base de données :

<https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager-webext/>

Comme il est possible aussi d'utiliser un de ces deux logiciels :

<https://sqlitebrowser.org/>

<https://sqlitestudio.pl/index.rvt>

The image shows two SQLite database management tools: DB Browser for SQLite and SQLiteStudio.

DB Browser for SQLite: This interface has a top menu bar with File, Edit, View, Tools, and Help. Below the menu is a toolbar with New Database, Open Database, Write Changes, Revert Changes, Open Project, Save Project, and Attach Database. The main area shows a table named "comments" with columns _id and comment. The data is as follows:

_id	comment
1	Very nice
2	Cool
3	Cool
4	Cool
5	Very nice
6	Cool
7	Cool
8	Hate it
9	Hate it

A modal window titled "Edit Database Cell" is open over the table, showing the value "1" in a large text input field. A status bar at the bottom indicates "Type of data currently in cell: Text / Numeric 1 char(s)".

SQLiteStudio: This interface has a top menu bar with Database, Structure, View, Tools, and Help. Below the menu is a toolbar with various icons for database operations. The left pane shows the database structure with a tree view of tables and columns. The right pane shows the same table data in a grid view:

_id	comment
1	Very nice
2	Cool
3	Cool
4	Cool
5	Very nice
6	Cool
7	Cool
8	Hate it
9	Hate it



Création d'interfaces utilisateur avancées



Interfaces avancées

- *Android offre des possibilités en ergonomie pour capter l'attention de vos utilisateurs.*
- I. Ce cours prolonge le cours sur les interfaces utilisateur, en montrant comment créer des **interfaces beaucoup plus riches**, notamment en créant **des vues personnalisées**.
 - II. Les **adaptateurs** permettront la réalisation d'interfaces orientées données en associant une source de données à des contrôles d'interface, par exemple pour **créer une liste de contacts ou de pays**.



Interfaces avancées

III. Ce cours traitera également de **l'animation des vues**, pour plus de **dynamisme visuel**, ainsi que de la création de **gadgets**.

IV. **L'internationalisation** des applications est également un point clé de la réussite de vos applications auprès des utilisateurs – tant pour leur diffusion que leur adoption. C'est ce que permet Android, en facilitant **l'adaptation de l'application à la langue de l'utilisateur**.



I. Créer des composants d'interface personnalisés



Les widgets, ou vues standards

- Si les différentes **vues standards** ne suffisent plus à réaliser tous vos projets, si vous avez des besoins plus complexes en matière d'interface utilisateur ou avec un style graphique radicalement différent, vous n'aurez d'autre choix que de les **créer vous-même**.



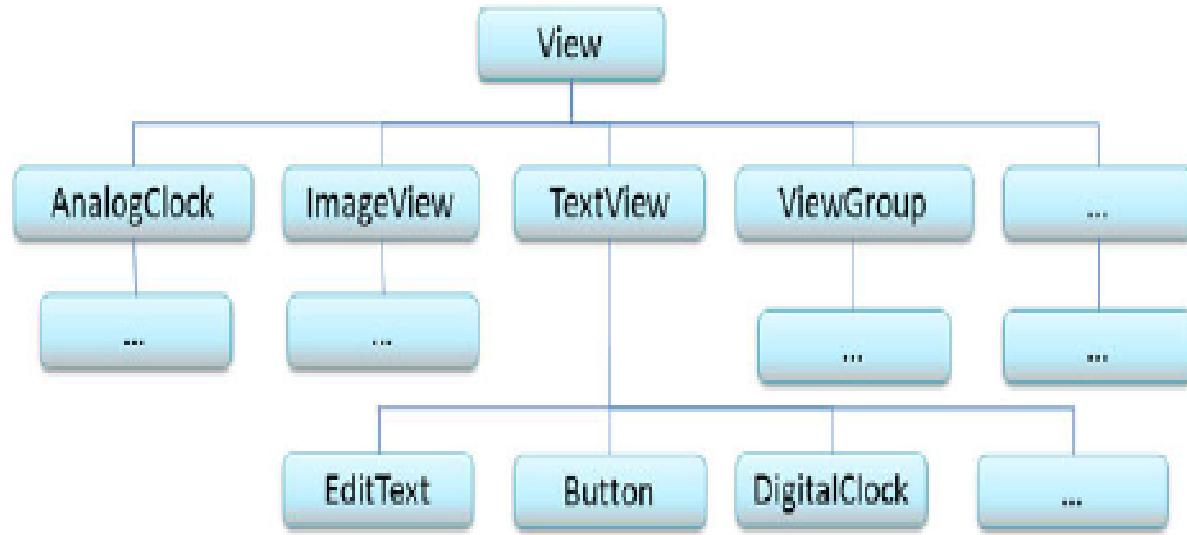
Les widgets, ou vues standards

- Le mot **widget** désigne l'ensemble des vues standards incluses dans la plate-forme Android. Elles font partie du paquetage **android.widget**.
- Les vues héritant toutes de la classe **View**, chaque widget hérite aussi de cette classe.
- Ainsi l'élément **Button** hérite-t-il de **TextView**, qui lui-même hérite de classe **View**. L'élément **CheckBox**, quant à lui, hérite de la vue **Button**.
- Android tire profit **des méthodes de chaque vue** et de chaque widget pour former une plate-forme **paramétrable** et **modulaire**.



Les widgets, ou vues standards

Figure 5–1 Structure de l'héritage des contrôles





Les widgets, ou vues standards

Voici une liste non exhaustive des contrôles actuellement disponibles :

- `Button` : un simple bouton poussoir ;
- `CheckBox` : contrôle à deux états, coché et décoché ;
- `EditText` : boîte d'édition permettant de saisir du texte ;
- `TextView` : contrôle de base pour afficher un texte ;
- `ListView` : conteneur permettant d'afficher des données sous forme de liste ;
- `ProgressBar` : affiche une barre de progression ou une animation ;
- `RadioButton` : bouton à deux états s'utilisant en groupe.

- Vous pouvez décrire une interface de deux façons :
- soit via une définition XML,
- Soit directement depuis le code en instanciant directement les objets adéquats.
- La construction d'une interface au sein du code d'une activité, plutôt qu'en utilisant une définition XML, permet par exemple de créer des interfaces dynamiquement.



Créer un contrôle personnalisé

Code 5-1 : Instantiation d'un Widget au sein d'une activité

```
Button button = new Button(this);
button.setText("Cliquez ici");
button.setOnClickListener(this);
...
```

- Il peut arriver qu'une vue ne réponde pas totalement aux besoins du développeur.
- Dans ce cas il est possible, au même titre que les autres widgets d'hériter d'une vue existante pour former un contrôle personnalisé.



Créer un contrôle personnalisé

- Pour créer un contrôle personnalisé, **la première étape** est de déterminer quelle est **la classe la plus adaptée** pour former la base de votre contrôle.
- Dans l'exemple suivant, nous allons créer un contrôle affichant **une grille**.

Nous décidons d'étendre la vue de base **TextView** pour afficher cette grille :



Code 5-2 : Création d'un composant personnalisé : classe CustomView

```
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.widget.TextView;

public class CustomView
    extends TextView {

    private Paint mPaint;
    private int mColor;
    private int mDivider;

    // Constructeurs
    public CustomView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        init(context);
    }

    public CustomView(Context context, AttributeSet attrs) {
        super(context, attrs);
        init(context);
    }

    public CustomView(Context context) {
        super(context);
        init(context);
    }

    // Initialisation du contrôle
    private void init(Context context){
        mDivider = 10;
        mColor = 0xff808080;
        mPaint = new Paint();
        mPaint.setColor(mColor);
    }
}
```



```
// Écriture de la propriété "GridColor"
public void setGridColor(int color){
    mColor = color;
    mPaint.setColor(mColor);
    invalidate();
}

// Lecture de la propriété "GridColor"
public int getGridColor(){
    return mColor;
}

// Écriture de la propriété "Divider"
public void setDivider(int divider){
    mDivider = divider;
    invalidate();
}

// Lecture de la propriété "Divider"
public int getDivider(){
    return mDivider;
}

@Override
protected void onDraw(Canvas canvas) {
    // On récupère la taille de la vue
    int x = getWidth();
    int y = getHeight();
    int n = x / mDivider;
    // On dessine des lignes verticales
    for(int i = 0; i<x ; i+=n){
        canvas.drawLine(i, 0, i, y, mPaint);
    }
    n = y / mDivider;
    // On dessine des lignes horizontales
    for(int j = 0; j<y ; j+=n){
        canvas.drawLine(0, j, x, j, mPaint);
    }
    // On dessine en dessous du texte
    super.onDraw(canvas);
}
```



Créer un contrôle personnalisé

- Vous noterez la déclaration des trois constructeurs de base d'une vue. Ils servent notamment à créer une vue à partir d'un gabarit d'interface au format XML. De cette façon, la création du contrôle et son utilisation dans un fichier de définition XML sont semblables à tout autre contrôle de la plate-forme Android.
- Vous remarquerez également que le code 5-2 contient quelques méthodes supplémentaires **set/get** permettant respectivement de mettre à jour et de récupérer les propriétés du contrôle. Les deux propriétés ajoutées sont **la couleur de grille** et **le nombre de cellules**.
- Pour mettre à jour l'affichage du contrôle nous emploierons conjointement les méthodes **invalidate** qui à son tour déclenchera un appel à la méthode **onDraw**. Le système nous fournira un **Canvas** sur lequel nous pourrons dessiner la grille de notre contrôle.



Créer un contrôle personnalisé

- Afin d'éprouver le contrôle personnalisé que vous venez de créer, construisez une activité et définissez le contrôle comme contenu principal. Vous pourrez utiliser l'écran tactile pour changer la taille des cellules de la grille :



Code 5-3 : Utilisation de notre contrôle dans du code

Créé

alisé

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

public class MainActivity
    extends Activity
{

    private CustomView view;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        view = new CustomView(this);
        view.setText("Mon contrôle");
        view.setGridColor(0xff800080);
        setContentView(view);
    }

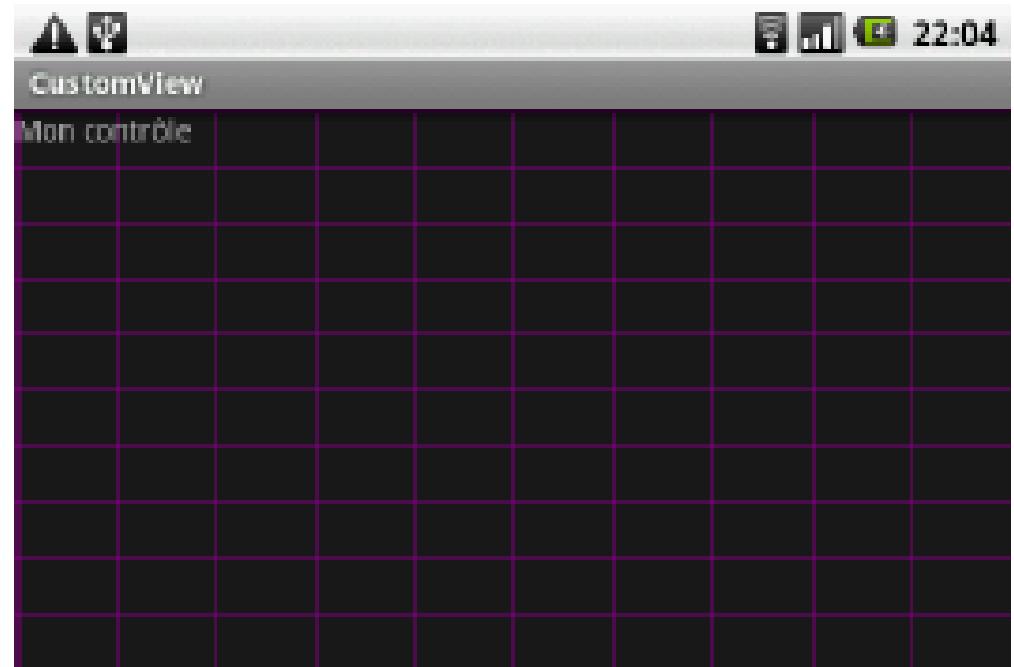
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        switch(event.getAction()){
            case MotionEvent.ACTION_DOWN:
                view.setDivider(20);
                break;

            case MotionEvent.ACTION_UP:
                view.setDivider(10);
                break;
        }
        return super.onTouchEvent(event);
    }
}
```



Créer un contrôle personnalisé

Figure 5–2
Rendu de notre contrôle





Déclarer un contrôle personnalisé dans les définitions d'interface XML

- Créer une interface utilisateur directement dans le code d'une activité n'est pas une bonne pratique (même si cela peut se révéler nécessaire pour certains scénarios). Nous avons abordé l'utilisation des contrôles standards d'Android dans les définitions XML des interfaces des activités.
- Cette méthode a plusieurs avantages : séparation entre la logique fonctionnelle et la présentation, interface modifiable par un graphiste et non uniquement par un développeur, etc.



Déclarer un contrôle personnalisé dans les définitions d'interface XML

- Contrairement à l'utilisation des contrôles Android standards, l'utilisation d'un contrôle personnalisé dans un fichier de description d'interface XML nécessite un peu de préparation en amont.
- En effet, pour pouvoir renseigner les propriétés d'un contrôle personnalisé d'une interface de manière déclarative, vous devez tout d'abord exposer ses attributs dans un fichier **res/attrs.xml** :



Déclarer un contrôle personnalisé dans les définitions d'interface XML

Code 5-4 : Fichier attrs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="CustomView">
        <attr name="GridColor" format="color"/>
        <attr name="Divider">
            <enum name="big" value="3" />
            <enum name="normal" value="10" />
            <enum name="small" value="20" />
        </attr>
    </declare-styleable>
</resources>
```

- Une fois les propriétés déclarées dans le fichier res(attrs.xml, celles-ci seront utilisables dans un fichier XML de description d'interface en ajoutant le nom du paquetage dans l'espace de nom comme indiqué ci-dessous :



Déclarer un contrôle personnalisé dans les définitions d'interface XML

Code 5-5 : Fichier main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res/
        com.eyrolles.android.customview"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <com.eyrolles.android.customview.CustomButton
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:id="@+id/CustomViewId"
        android:text="Cliquez ici"
        app:GridColor="#ffff0000"
        app:Divider="big"
        />
</LinearLayout>
```



Déclarer un contrôle personnalisé dans les définitions d'interface XML

- Après avoir défini les propriétés dans un fichier d'attributs et avant de pouvoir profiter pleinement de votre contrôle personnalisé en mode déclaratif,
- il faut revoir les constructeurs contenus dans le code du contrôle personnalisé. En effet, chaque propriété personnalisée sera passée en paramètre du constructeur sous la forme d'un tableau d'objets.
- Vous devez donc manipuler ce tableau pour récupérer les valeurs de ces propriétés et les appliquer au contrôle personnalisé :



Code 5-6 : Chargement des propriétés depuis le fichier xml

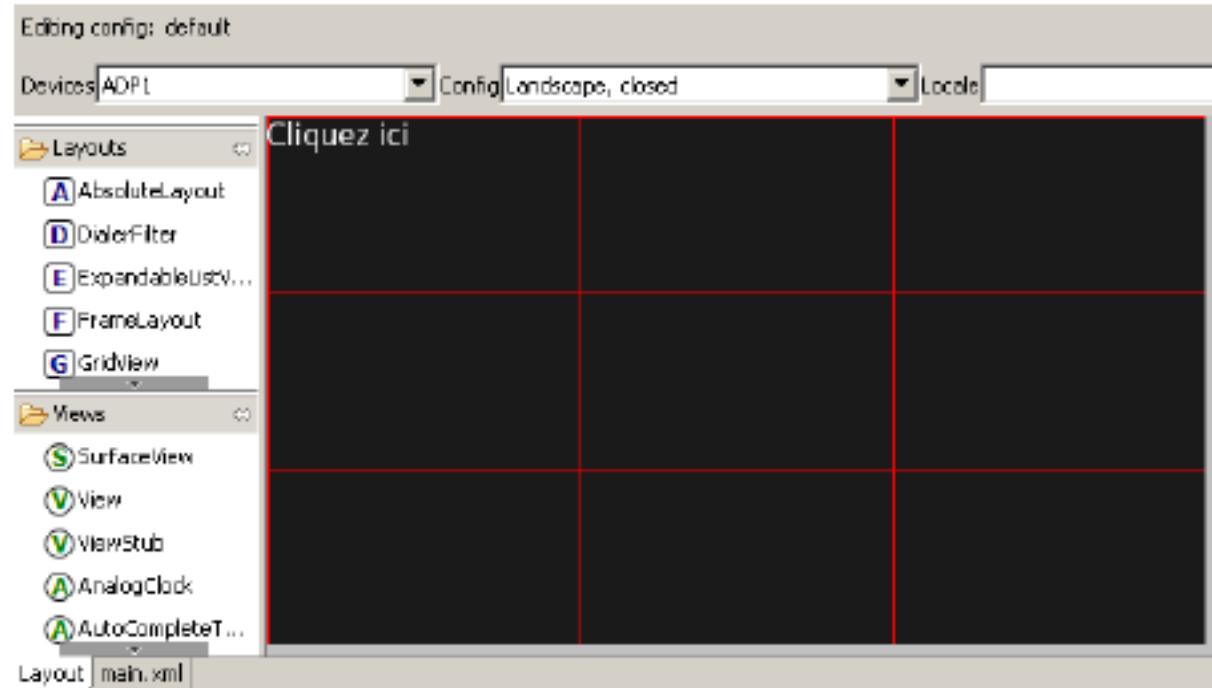
```
public CustomView(Context context, AttributeSet attrs, int defStyle) {  
    super(context, attrs, defStyle);  
    mDivider = 10;  
    mColor = 0xff808080;  
    mPaint = new Paint();  
    loadAttrFromXml(context, attrs, defStyle);  
    mPaint.setColor(mColor);  
}  
  
public CustomView(Context context, AttributeSet attrs) {  
    this(context, attrs, 0);  
}  
  
public CustomView(Context context) {  
    this(context, null);  
}  
  
// Charge les propriétés à partir du fichier xml  
private void loadAttrFromXml(Context context, AttributeSet attrs, int defStyle){  
    TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.CustomButton,  
                                                defStyle, 0);  
    int n = a.getIndexCount();  
    for (int i = 0; i < n; i++) {  
        int attr = a.getIndex(i);  
        switch (attr) {  
            case R.styleable.CustomButton_Divider:  
                mDivider = a.getInt(attr, mDivider);  
                break;  
            case R.styleable.CustomButton_GridColor:  
                mColor = a.getColor(attr, mColor);  
                break;  
        }  
    }  
    a.recycle();  
}
```



Déclarer un contrôle personnalisé dans les définitions d'interface XML

Figure 5–3

Rendu du contrôle dans le concepteur d'interface d'ADT(voir annexes).





II. Les adaptateurs pour accéder aux données de l'interface.



Les adaptateurs

- Les adaptateurs servent à gérer les sources de données qui seront notamment affichées dans les différentes listes de l'interface utilisateur.
- Ils réalisent la liaison entre vos sources de données (un simple tableau de chaînes de caractères, une base de données, un fournisseur de contenu, etc.) et les contrôles de votre interface utilisateur.



Les adaptateurs

Figure 5–4
L'objet Adapter dans
son environnement





Les adaptateurs

- Prenons comme exemple un tableau de chaînes de caractères à afficher sous la forme d'une liste dans une activité.
- Commençons par déclarer **le tableau** puis un **ArrayAdapter**. Pour simplifier la gestion des vues, nous appliquerons cet adaptateur dans une activité de type **ListActivity**.
- Ce type d'activité embarque en effet une **ListView** et des méthodes simplifiant la gestion des adaptateurs.



Les adaptateurs

Code 5-8 : Création d'un adaptateur

```
String[] tableau = new String[]{
    "Un" , "Deux" , "Trois" , "Quatre"
    , "Cinq" , "Six" , "Sept" , "Huit"
    , "Neuf" , "Dix"};
```

```
ArrayAdapter<String> adapter =
new ArrayAdapter<String>(
    this
    , android.R.layout.simple_list_item_1, tableau);
this.setAdapter(adapter);
```

Figure 5-5
Exemple de liste





Les adaptateurs

Le constructeur utilisé pour instancier un nouvel objet ArrayAdapter prend trois paramètres :

- **Context context** : grâce à ce paramètre l'adaptateur sera capable de créer seul les objets nécessaires à la transcription des fichiers XML en ressources ;
- **int textViewResourceId** : l'identifiant du fichier XML à utiliser comme modèle pour la liste (en général contenu dans le dossier /res/layout du projet). Le modèle est la vue ou le groupe de vue qui sera utilisé par chaque entrée pour s'afficher dans la liste associée avec l'ArrayAdapter ;
- **String[] objects** : les données sous forme d'un tableau de chaînes de caractères.



III. Animation des vues



Animation des vues

- Les animations font aujourd’hui partie de notre quotidien que ce soit sur le Web (via les technologies XHTML/CSS/JavaScript, Adobe Flash/Flex, Microsoft Silverlight, etc.) ou dans les applications bureautiques. Face à cela, la plate-forme Android n’est pas en reste puisqu’elle propose deux mécanismes pour animer vos interfaces :
- **les animations image par image** (dédiées comme son nom l’indique, aux images) ;
- **les animations de positionnement** (dédiées à l’animation des vues).



Animation des vues

- La plate-forme Android permet là encore de définir des animations des deux façons habituelles : par le code ou via l'utilisation d'un fichier XML.

- Définir les animations dans des fichiers de ressources externes offre deux atouts : d'une part, une personne de l'art tel qu'un designer peut modifier le fichier indépendamment du code et d'autre part, Android peut alors appliquer automatiquement l'animation la plus appropriée selon le matériel (taille et orientation de l'écran).



Les animations d'interpolation

- Les animations d'interpolation permettent d'effectuer une rotation et/ou de changer la position, la taille, l'opacité d'une vue.
- Avec ce type d'animation, la plate-forme s'occupera pour vous de définir les étapes intermédiaires par interpolation en fonction du temps et des états d'origine et de fin que vous aurez spécifiés.
- Ce type d'animation est souvent utilisé pour réaliser des transitions entre activités ou pour mettre en exergue un élément de l'interface à l'utilisateur (saisie incorrecte, options à la disposition de l'utilisateur, etc).



Les animations d'interpolation

- Les interpolations possibles sont les suivantes :
 - **opacité (Alpha)** : permet de jouer sur la transparence/opacité de la vue ;
 - **échelle (Scale)** : permet de spécifier l agrandissement/réduction sur les axes et X et Y à appliquer à la vue ;
 - **translation (Translate)** : permet de spécifier la translation/déplacement à effectuer par la vue ;
 - **rotation (Rotate)** : permet d'effectuer une rotation selon un angle en degré et un point de pivot.



Animer par le code

- Chaque type d'animation par interpolation possède une sous-classe dérivant de Animation : **ScaleAnimation**, **AlphaAnimation**, **TranslateAnimation** et **RotateAnimation**. Chacune de ces animations peut être créée, paramétrée, associée et exécutée directement depuis le code :

Code 5-31 : Animer une vue directement depuis le code

```
TextView vueTexte = ...;

// Animation d'une alpha
Animation animationAlpha = new AlphaAnimation(0.0f, 1.0f);
animationAlpha.setDuration(100);
vueText.startAnimation(animationAlpha);
```



```
// Animation de translation d'une vue
TranslateAnimation animationTranslation = new TranslateAnimation(
    Animation.RELATIVE_TO_SELF, 0.0f, Animation.RELATIVE_TO_SELF, 0.0f,
    Animation.RELATIVE_TO_SELF, -1.0f, Animation.RELATIVE_TO_SELF, 0.0f);
animationTranslation.setDuration(500);
vueText.startAnimation(animationTranslation);
```

- Chaque type d'animation possède des constructeurs propres prenant des paramètres qui leur sont spécifiques. D'une manière globale, la plupart des méthodes sont communes.
- Par exemple, pour spécifier la durée de l'animation, utilisez la méthode **setDuration** spécifiant le nombre de millisecondes en paramètre.
- Enfin, pour démarrer une animation, appelez la méthode **startAnimation** de la vue cible et spécifiez l'objet Animation que vous souhaitez utiliser.
- Nous verrons plus loin comment créer une série d'animation par le code, c'est-à-dire un enchaînement d'animations élémentaires.



Animer par le XML

- Le tableau suivant présente les attributs XML des paramètres de chaque type d'animation(vous remarquerez que ces propriétés XML retrouvent toutes un équivalent sous la forme de méthodes d'accès **get/set** dans les classes dérivées d'Animation).

Tableau 5–3 Les propriétés des animations par interpolation

Type d'animation	Propriété	Description
ScaleAnimation	fromXScale	float
	toXScale	float
	fromYScale	float
	toYScale	float
	pivotX / pivotY	Valeur en pourcentage du centre de rotation, ou pivot, relatif à la vue sur l'axe des ordonnées. Chaîne dont la valeur est comprise entre '0 %' et '100 %'.
AlphaAnimation	fromAlpha	Valeur de l'opacité de départ. Valeur comprise entre 0.0 et 1.0 (0.0 est transparent).
	toAlpha	Valeur de l'opacité d'arrivée. Valeur comprise entre 0.0 et 1.0 (0.0 est transparent).
TranslateAnimation	fromX	float
	toX	float

Type d'animation	Propriété	Description
RotateAnimation	fromDegrees	Orientation de départ en degrés. Valeur comprise entre 0 et 360.
	toDegrees	Orientation d'arrivée en degrés. Valeur comprise entre 0 et 360.
	PivotX / pivotY	Valeur en pourcentage du centre de rotation, ou pivot, relatif à la vue sur l'axe des abscisses. Chaîne dont la valeur est comprise entre '0 %' et '100 %'.

- Les fichiers XML de description d'animation doivent être placés dans le répertoire **res/anim** afin que le système génère un identifiant qui sera ensuite utilisé pour y faire référence dans le code.



➤ L'extrait de fichier XML suivant présente la définition d'animations de chaque type:

Code 5-32 : Ensemble d'animations

```
// Une animation sur l'opacité de la vue.  
<alpha  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:interpolator="@android:anim/accelerate_interpolator"  
    android:fromAlpha="0.0"  
    android:toAlpha="1.0"  
    android:duration="1000" />  
  
// Animation de translation  
<translate android:fromXDelta="100%p"  
    android:toXDelta="0"  
    android:duration="1000"/>  
  
// Animation d'échelle  
<scale android:fromXScale="0.0"  
    android:toXScale="1.0"  
    android:fromYScale="0.0"  
    android:toYScale="1.0"  
    android:pivotX="50%"  
    android:pivotY="50%"  
    android:duration="1000" >  
  
// Animation de rotation  
<rotate android:fromDegrees="0"  
    android:toDegrees="360"  
    android:pivotX="50%"  
    android:pivotY="50%"  
    android:duration="1000" >
```



- Une fois votre animation décrite, vous devrez charger et associer l'animation à une vue – ou un groupe de vues – afin d'animer celle-ci. Le chargement s'effectue à l'aide de la méthode statique **loadAnimation** de la classe **AnimationUtils** en spécifiant l'identifiant de la ressource d'animation.
- Pour terminer, associez l'animation ainsi chargée à la vue en appelant sa méthode **setAnimation** et en spécifiant l'objet Animation précédemment chargé à partir de la définition XML de l'animation.

Code 5-33 : Charger et exécuter une animation décrite en XML

```
Animation animation = AnimationUtils.loadAnimation(this, R.anim.animation1);
animation.setRepeatMode(Animation.RESTART);
animation.setRepeatCount(Animation.INFINITE);
maVueAAnimer.startAnimation(animation);
```



Créer une série d'animations

- Vous pouvez créer une série d'animations en utilisant la classe **AnimationSet** ou l'attribut `set` dans la définition XML. Vous allez pouvoir exécuter ces animations soit simultanément soit en décalé dans le temps.

- Pour vous aider à planifier l'exécution des différentes animations vous pouvez utiliser les attributs XML (ou leur méthode équivalente au niveau des classes) communs à tous les types d'animations par interpolation.



Créer une série d'animations

Tableau 5–4 Propriétés communes des animations par interpolation pour l'exécution planifiée

Nom de la propriété	Description
duration	La durée en millisecondes de l'exécution totale de l'animation.
fillAfter	Une valeur booléenne pour spécifier si la transformation doit s'effectuer avant l'animation.
fillBefore	Une valeur booléenne pour spécifier si la transformation doit s'effectuer après l'animation.
interpolator	L'interpolateur utilisé pour gérer l'animation (cf. ci-après dans ce chapitre).
startOffset	Le délai en millisecondes avant le démarrage de l'animation. La référence à l'origine de ce délai est le début de la série d'animation. Si vous ne spécifiez pas cette propriété, l'animation sera exécutée immédiatement, de la même façon que vous auriez spécifié la valeur 0.
repeatCount	Définit le nombre de fois où l'animation doit être jouée.
repeatMode	Définit le mode de répétitions si le nombre de répétitions est supérieur à 0 ou infini. Les valeurs possibles sont <code>restart</code> (valeur 1) pour recommencer l'animation, et <code>reverse</code> (valeur 2) pour jouer l'animation à l'envers.



Code 5-34 : Réalisation d'une série d'animation en code Java

```
// Création d'une série d'animations. La valeur true spécifie que
// chaque animation utilisera l'interpolateur de l'AnimationSet.
// Si false, chaque animation utilisera l'interpolateur
// défini dans chacune d'elle.
AnimationSet serieAnimations = new AnimationSet(true);

AlphaAnimation animationAlpha = new AlphaAnimation(0.0f, 1.0f);
animationAlpha.setDuration(500);
// Ajout de l'animation à la série d'animation
serieAnimation.addAnimation(animation);

TranslateAnimation animationTranslation = new TranslateAnimation(
    Animation.RELATIVE_TO_SELF, 0.0f,
    Animation.RELATIVE_TO_SELF, 0.0f,
    Animation.RELATIVE_TO_SELF, -1.0f,
    Animation.RELATIVE_TO_SELF, 0.0f
);
animationTranslation.setDuration(300);
animationTranslation.setStartOffset(500);
// Ajout de l'animation à la série d'animations
serieAnimation.addAnimation(animation);

// Exécution de la série d'animations
vueAAnimer.startAnimation(serieAnimation);
```



Code 5-35 : Réalisation d'une série d'animation en XML

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
      android:interpolator="@android:anim/accelerate_interpolator">
    <alpha
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:fromAlpha="0.0" android:toAlpha="1.0" android:duration="500" />
    <translate android:fromXDelta="0"
              android:toXDelta="100%p"
              android:startOffset="500"
              android:duration="300" />
</set>
```



Animer un groupe de vues

- Animer une vue permet de créer des effets très saisissants mais limités à leur périmètre.
- Si vous souhaitez animer la page complète, par exemple pour réaliser une page qui se tourne ou un effet d'apparition digne des transitions des meilleurs logiciels de présentation, Android permet d'appliquer une transformation à une mise en page complète.
- Pour animer un groupe de vues (toute mise en page dérivant de la classe **ViewGroup**) vous devrez spécifier l'animation à la méthode **setLayoutAnimation** du groupe de vues.



Animer un groupe de vues

Code 5-36 : Chargement et exécution d'une animation d'un groupe de vues

```
AnimationSet serieAnimations = new AnimationSet(true);
...
serieAnimation.addAnimation(animation);

TranslateAnimation animationTranslation = ...
...
serieAnimation.addAnimation(animation);

// Crée un gestionnaire d'animation qui appliquera l'animation à
// l'ensemble des vues avec un délai - ici nul - entre chaque vue.
LayoutAnimationController controller =
        new LayoutAnimationController(serieAnimation, 0.f);
// Spécifie l'animation de groupe de vues à utiliser.
groupeVues.setLayoutAnimation(controller);
// Démarré l'animation du groupe de vues.
groupeVues.startLayoutAnimation();
```



Les animations image par image

- Les animations image par image permettent de spécifier une séquence d'images qui seront affichées les unes à la suite des autres selon un délai spécifié, un peu à la manière d'un dessin animé.

- Comme les animations par interpolation, une séquence d'animation image par image peut être définie soit dans le code soit dans une ressource XML externe. La séquence d'images est décrite dans une liste d'éléments item contenant dans un élément parent animation-list. Pour chaque élément, vous spécifiez l'identifiant de la ressource et sa durée d'affichage :



Les animations image par image

Code 5-41 : Séquence d'une animation image par image

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    <!-- Ce paramètre est utilisé pour spécifier si l'animation doit être
        jouée une ou plusieurs fois -->
    android:oneshot="false">
    <!-- Chaque image est déclarée dans un élément item -->
    <item android:drawable="@drawable/android1" android:duration="500" />
    <item android:drawable="@drawable/android2" android:duration="500" />
    <item android:drawable="@drawable/android3" android:duration="500" />
</animation-list>
```



Les animations image par image

- Au niveau du code vous devez spécifier l'animation à la vue via sa méthode `setBackgroundResource` en précisant l'identifiant de l'animation séquentielle d'images.
- Pour démarrer l'animation, récupérez l'objet `AnimationDrawable` en appelant la méthode `getBackground` de la vue, puis appelez la méthode `start` de l'animation.

À SAVOIR Ne pas démarrer l'animation dans la méthode `onCreate`

Attention à l'endroit où vousappelez la méthode `start`. En effet, si vousappelez cette méthode pour lancer votre animation depuis la méthode `onCreate`, l'animation nedémarrera pas !



Les animations image par image

Code 5-42 : Animation d'une vue avec une animation image par image

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
  
    ImageView image = (ImageView) findViewById(R.id.image);  
    // Spécifie l'animation comme fond de l'image  
    image.setBackgroundDrawable(getResources().getDrawable(R.drawable.animation_images));  
    image.setOnClickListener(new View.OnClickListener() {  
  
        @Override  
        public void onClick(View v) {  
            ImageView image = (ImageView) AnimationImageParImage.this  
                .findViewById(R.id.image);  
            // Récupère l'animation à animer  
            AnimationDrawable animation = (AnimationDrawable) image  
                .getBackground();  
            // Démarre l'animation  
            animation.start();  
        }  
    });  
}
```



Les animations image par image

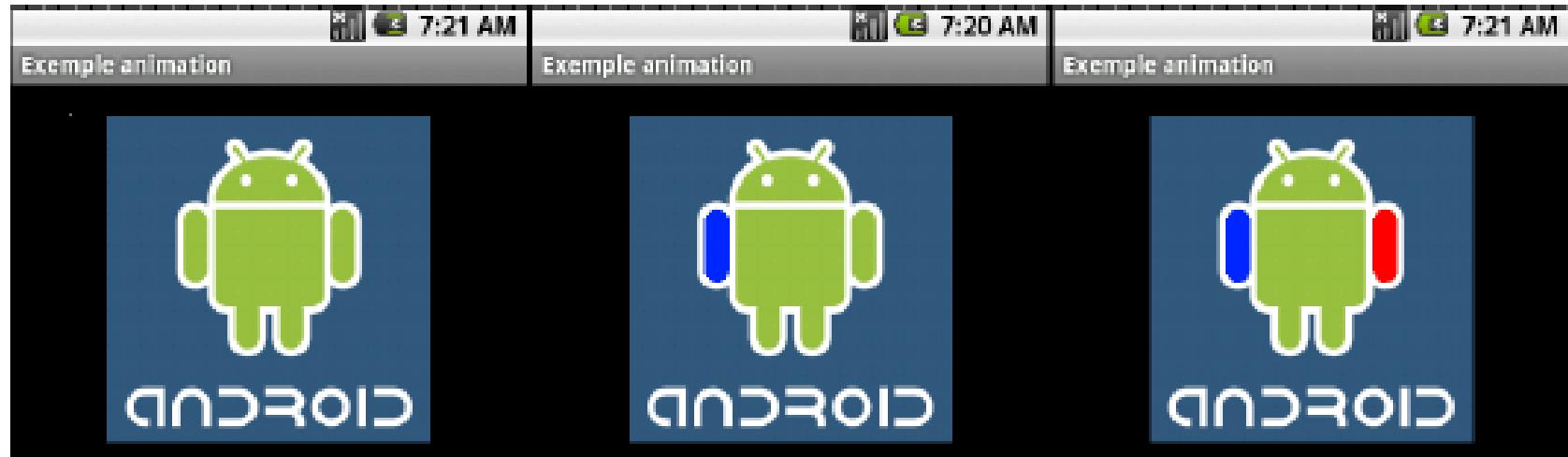


Figure 5–25 Résultat de l'animation du code 5-42



Les AppWidgets

- Hormis les icônes des applications, le bureau (ou écran d'accueil) peut également contenir ce que l'on appelle les **AppWidgets**. Leur utilisation consiste à exporter votre application sous la forme d'un contrôle personnalisable disponible directement sur le bureau de l'utilisateur.

- Les gadgets sont une très bonne manière d'étendre vos applications. Ils offrent un accès direct aux fonctionnalités de celles-ci, tout en gardant un format discret. Nous pouvons les comparer à de petits programmes embarqués, tels que des extensions, comme le proposent aujourd'hui les systèmes d'exploitation de dernière génération.



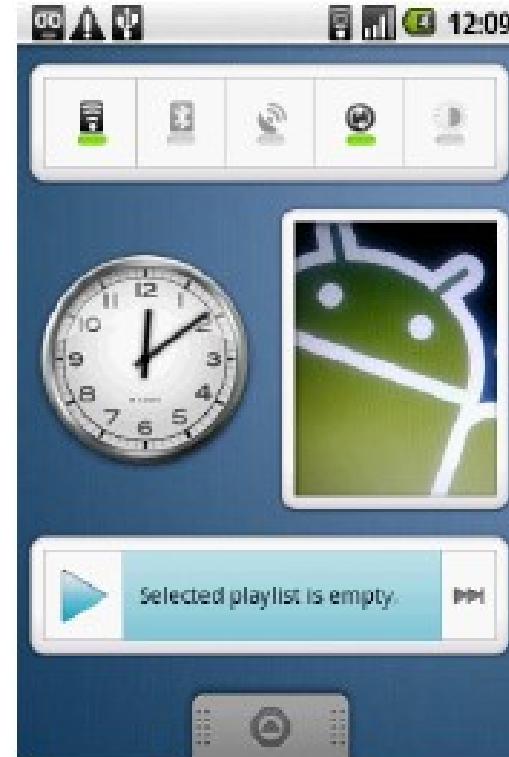
Les AppWidgets

- Les avantages sont nombreux :
 - il est inutile de lancer l'application à proprement parler ;
 - l'application est toujours disponible sur le bureau ;
 - plusieurs gadgets peuvent se partager un même écran ;
 - les ressources de fonctionnement ne sont allouées que lorsque cela est nécessaire ;
 - chaque gadget peut sauvegarder ses propres paramètres.
- À titre d'exemple l'horloge analogique fait partie des gadgets, ainsi que le lecteur média, l'album photo et le panneau de contrôle comme le montre l'écran ci-joint.



Les AppWidgets

Figure 5–26
Exemples de gadgets
(source Google)





Création d'un gadget

- Pour concevoir un gadget nous allons avoir besoin de réaliser plusieurs tâches :
 1. créer un objet hérité de la classe AppWidgetProvider ;
 2. définir l'aspect visuel du gadget dans un autre fichier XML ;
 3. définir les paramètres du AppWidgetProvider dans un fichier XML
 - 4 .ajouter une activité pour personnaliser le gadget ;
 - 5 .déclarer le gadget dans le fichier de configuration de l'application.



IV. Internationalisation des applications



Internationalisation des applications

- Pour que vos applications rencontrent le plus grand succès, il faut les rendre compréhensibles par le plus grand nombre d'utilisateurs, et donc les traduire. Les auteurs de la plate-forme Android ont bien pris en compte cette nécessité et offrent un mécanisme assez facile à prendre en main.



Étape 0 : créer une application à manipuler

Code 5-28 : Mise en page d'une application avec une image et une case à cocher

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ① xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<ImageButton ②
    android:id="@+id/android_button"
    android:layout_width="100dip"
    android:layout_height="wrap_content"
    android:src="@drawable/android" />

<CheckBox android:id="@+id/checkbox" ③
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/checkboxttext" />
</LinearLayout>
```

Cette mise en page est simple avec une disposition linéaire ①, une image du bonhomme Android ② et une case à cocher ③ contenant un texte à traduire.

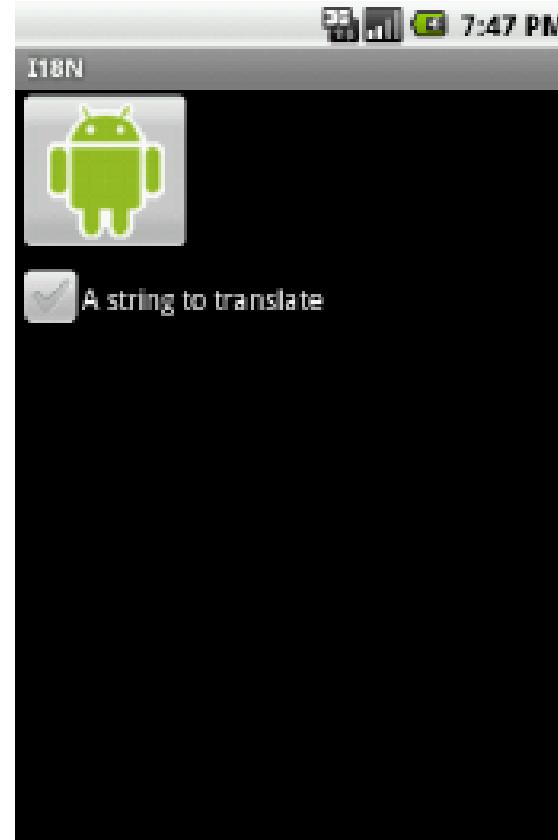
Il faut également renseigner le fichier ressource qui contient les chaînes de caractères.



Code 5-29 : Fichier strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">I18N</string>
    <string name="checkboxtext">A string to translate</string>
</resources>
```

Figure 5–19
Capture d'écran de
l'application sans traduction





Internationaliser des chaînes de caractères

Maintenant, commençons à personnaliser l'application en fonction de la langue de l'appareil. Cliquez sur l'icône de l'assistant de création de fichiers XML ou parcourez les menus *File > New > Android XML File*.

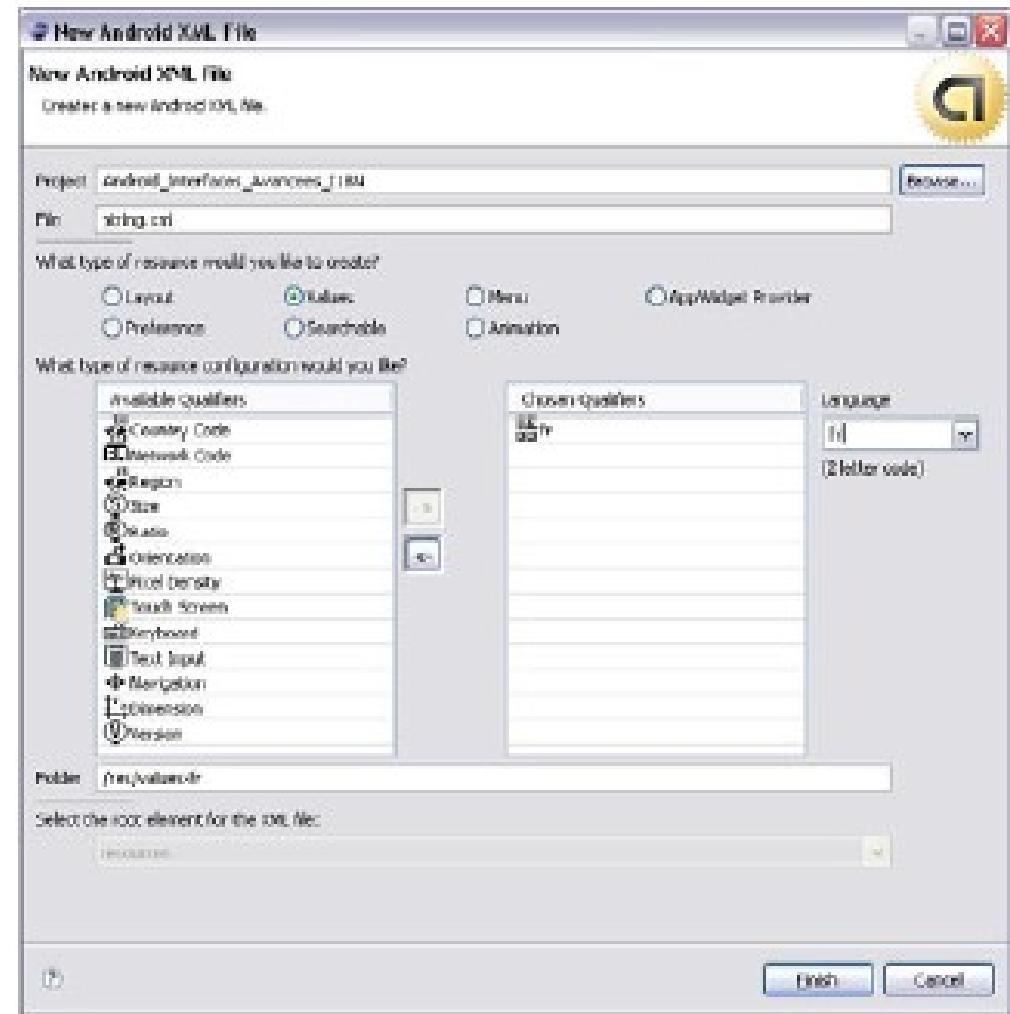
Figure 5–20
Icône de l'assistant de création
de fichiers XML



Grâce au formulaire qui apparaît, sélectionnez votre projet puis entrez un nom de fichier à traduire (`strings.xml` si vous n'avez pas modifié le projet). Spécifiez ensuite *Values* comme type de ressource à créer. Sélectionnez *Language* dans la liste gauche et appuyez sur la flèche. Ensuite dans la boîte *Language* qui doit apparaître, à droite de la fenêtre, saisissez le code à deux lettres qui correspond au langage que vous souhaitez utiliser. Prenons *fr* comme exemple.



Figure 5–21
Ajout d'un fichier de traduction
en français



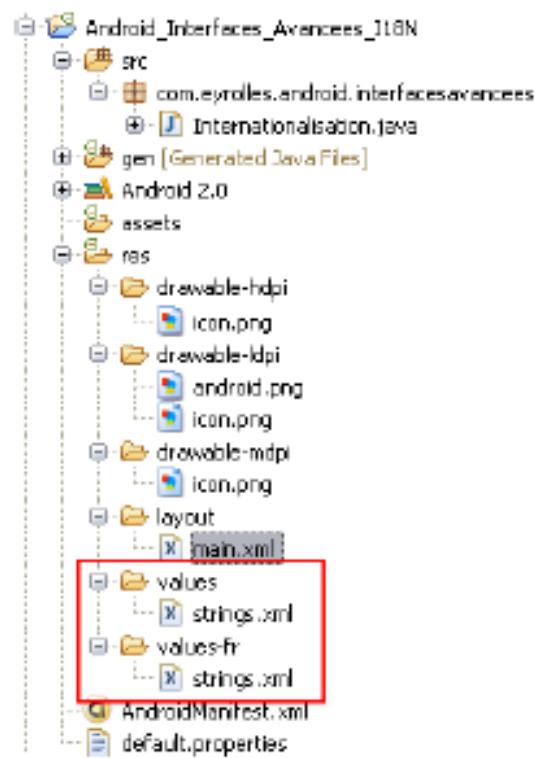
À CONSULTER Liste des langues et des pays

Vous pouvez consulter la liste des codes de langues et de pays aux adresses suivantes :

- ▶ http://www.loc.gov/standards/iso639-2/php/code_list.php
- ▶ http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm

Enfin, appuyez sur *Finish*. Votre projet devrait contenir un répertoire supplémentaire : `/res/values-fr`.

Figure 5–22
Impact sur le projet



Éditez le fichier strings.xml de ce nouveau répertoire et placez-y les chaînes traduites en français.

Code 5-30 : Chaînes traduites en français dans le répertoire res/values-fr

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">I18N France</string>
    <string name="checkboxtext">Une traduction pour la France</string>
</resources>
```

Ces modifications effectuées, relancez votre émulateur et vous devriez constater... que rien n'a changé ! En effet, cela est tout à fait normal étant donné que nous n'avons pas modifié la langue du système qui n'est pas le français par défaut.

Pour changer ce paramètre, ouvrez le volet des applications et sélectionnez *Custom Locale*. Le réglage affiché doit être *en_US*, sélectionnez *fr_FR* grâce à un appui long et à la validation demandée.



Figure 5–23
Réglage de la langue
du système



Relancez l'application, elle est bien en français !



Voici un tableau qui met en évidence la logique entre le code pays et les fichiers et répertoires nécessaires. Tous les codes pays possibles sont consultables grâce à l'application *Custom Locale*.

Tableau 5–1 Les différents pays avec leurs codes et l'emplacement de leurs ressources

Code pays	Pays/Langue concernés	Répertoire du fichier string.xml
Défaut	Angleterre/Anglais	res/values/
en-rUS	Etats-Unis/Anglais	res/values/
fr-rFR	France/Français	res/values-fr/
Ja-rJP	Japon/Japonnais	res/values-ja/
...



Internationaliser les images

Comme nous l'avons laissé entendre tout au long du chapitre, il est également possible de choisir des images différentes en fonction de la langue du système (utile par exemple si l'on veut afficher le drapeau de la langue courante).

Pour cela, créons une seconde image de la mascotte Android francisée, par exemple en lui rajoutant quelques couleurs. Il ne reste qu'à la placer au bon endroit. La logique est d'ailleurs similaire à celle qui dicte la gestion des chaînes de caractères.

Tableau 5–2 Les différents pays avec leurs codes et l'emplacement de leurs ressources images

Code pays	Pays/Langue concernés	Répertoire des images
Défaut	Angleterre/Anglais	res/drawable/
en-rUS	Etats-Unis/Anglais	res/drawable-en-rUS/
fr-rFR	France/Français	res/drawable-fr-rFR/
Ja-rJP	Japon/Japonnais	res/drawable-ja-rJP/
...

En consultant rapidement le tableau présenté ci-dessus, il faut placer cette nouvelle image – mais avec le même nom de fichier `android.png` – dans le répertoire `res/drawable-fr-FR`.

Relancez l'application et voilà, le tour est joué !



Figure 5–24
Résultat de notre
internationalisation

