Introduction

ANDROID

Un monde ouvert

- Première plateforme ouverte pour appareils mobiles.Les applications natives et tierces utilisent les mêmes API.
- Un système d'exploitation open source libre pour appareils mobiles;
- Un environnement de développement ouvert construit sur un noyau Linux open source;
- Très fascinant de par sa philosophie ouverte, Android offre la possibilité de corriger des interfaces utilisateurs natives ou la conception d'une application native (En écrivant une extension ou en procédant à un remplacement).

Concurrents

- Symbian OS: un système d'exploitation pour téléphone portable. Hérité de Epoc32, il est crée par différents constructeurs(Psion, Nokia, Motorola);

- iOS(iphone OS): Un système mobile développé par Apple;

- Windows mobile: C'est le nom générique donné à différentes versions de Microsoft Windows.

Architecture ANDROID

- Au niveau système, Android est organisé autour de plusieurs instances de la machine virtuel Dalvik, dans différents processus.

- Les instances communiquent entre elles à l'aide d'un mécanisme IPC (Inter Processus Call)

- Une instance particulière de Dalvik, portée par le processus SystemServer, expose des composants et des services aux applications

- 2 modes de lancement d'application
  1. Terminal monoprocess: 1 classloader / appli
  2. Terminal monoprocess: 1 process / appli

Architecture ANDROID

- Pour optimiser l'instanciation des machines virtuelles, Android utilise un processus spécifique apellé Zygote
- Zygote porte une première instance de Dalvik comme modèle.
- Zygote écoute sur un socket local.
- Sur réception d'une ligne de commande, celle-ci est interprétée par Zygote, après avoir effectué un fork() du processus.
- Zygote précharge une liste de classes en mémoire, lie les DLLs JNI et à faire le ménage à l'aide du ramasse-miettes.

Architecture ANDROID : code source

- Fichier source java : **SystemServer.java**
  - frameworks/base/services/java/com/android/server/
  - Appel de la librairie native android_servers qui fournit une interface aux fonctionnalités natives via JNI
    - frameworks/base/services/jni/

- Fichier source c++ : **system_init.cpp**
  - frameworks/base/cmds/system_server/library/

Architecture ANDROID

Android pour développeur

- Andoid est conçu par des développeurs pour des développeurs.
- Andoid nécessite aucune certification pour devenir développeur.
- Avec Android on peut distribuer et monétiser des applications sur Android Market.
- Puissant et intuitif, Andoid facilite le développement mobile.
- Ouvert, un SDK simple et puissant. L'absence de coût de licence attire plus de développeur.

Andoid pour constructeur

- Facilité d'accès au matériel de bas niveau avec une serie d'API disponible.
- Android fonctionne sur plusieurs marques de smartphones.
- L'interaction entre applications est transparente.
- Intégration de Google Maps et utilisation de services d'arrière plan dans les applications.
- Les applications natives et tierces utilisent les mêmes API.

Android et Java

- Android ressemble fortement à Java mais il en est pas. Les applications Android sont écrites en Java mais ne sont pas exécutées par une machine virtuelle Java ME.

- Les applications Java ne fonctionnent pas nativement sous Android.

- Les applications Android sont exécutées par une machine virtuelle spécifique Dalvik et non par une JVM classique.

Android et Java

- Android inclus une implémentation Open source spécifique de java basée non pas sur OpenJdk, mais sur le projet Harmony (implémentation java 5 de la fondation apache).

- Google a adapté Harmony et supprimé certains packages non nécessaire en environnement mobile.

Android et Java

- Packages.

Android et C

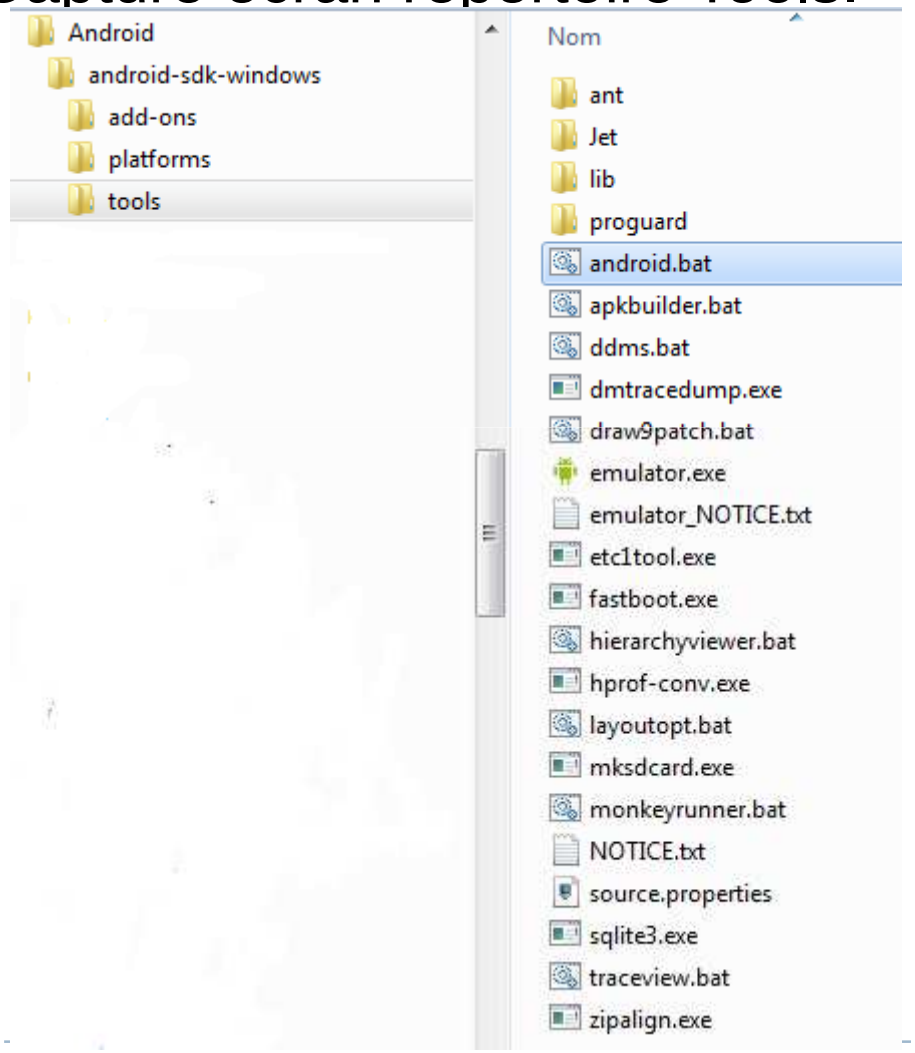
- Avec Android, il est possible d'écrire des applications en C/C++ qui sont exécutées directement par le système d'exploitation Linux.
- Android fournit un kit de développement natif (NDK).
- Ce NDK permet de créer des bibliothèques C++ à l'aide des bibliothèques libc et libm et donne un accès natif à OpenGL (Open Graphics Library)
- OpenGL est une spécification qui définit une API multiplate forme pour la conecption des applications générant des images 3D ou 2D.

Android et Linux

- Android est un système d'exploitation basé sur Linux noyau 2.6.
- Android issu d'un fork de la 2.6.x, mais n'est pas une distribution linux car il manque plusieurs composants comme X11 (graphique) et glibc
- La machine Dalvik utilise le noyau Linux pour gérer les fonctionnalités de bas niveau y compris la sécurité, le threading et la gestion des processus et de la mémoire.

Android et Lunix

- Capture écran répertoire Tools.



Boîte à outils

SDK Android

- Le kit de développement Android(SDK) fournit l'environnement de travail pour développer, tester et déboguer des applications Android.

- Dans le SDK on trouve:
  1. **Les API Android** qui sont le coeur du SDK. Composés de bibliothéques d'API Android, ils donnent au développeur accès à la pile Android.
  2. **Des outils de développement** qui permettent de compiler et déboguer vos applications.
  3. **Le virtual Device Manager et l'Emulateur** qui fournit un meilleur environnement de test.
  4. **Des exemples de code et un support en ligne**.

SDK Android

- Le SDK Android est disponible en téléchargement pour les plateformes Lunix, Mac et Windows à l'adresse suivante: **http://developer.android.com/skd/index.html**

- Il existe plusieurs versions du SDK. Chaque nouvelle version donne de nouvelle fonctionnalité. Exemple(la version du SDK 2.2 donne la possibilité d'installer les applications sur la carte SD. Elle contient aussi un back up manager pour sauvegarder les paramétres des applications).

Plugin Eclipse ADT

- Le plugin eclipse ADT simplifie le développement Android. Il intégre les outils de développement comme l'émulateur et le convertisseur .class-to-.dex

- Il fournit un assistant de projet Android qui permet de créer rapidement de nouveaux projets.

- Il automatise et simplifie le processus de construction des applications Android.

- Il fournit un éditeur qui aide à écrire du code XML valide pour le manifeste Android (AndroidManifest.xml) et les fichiers de resources.

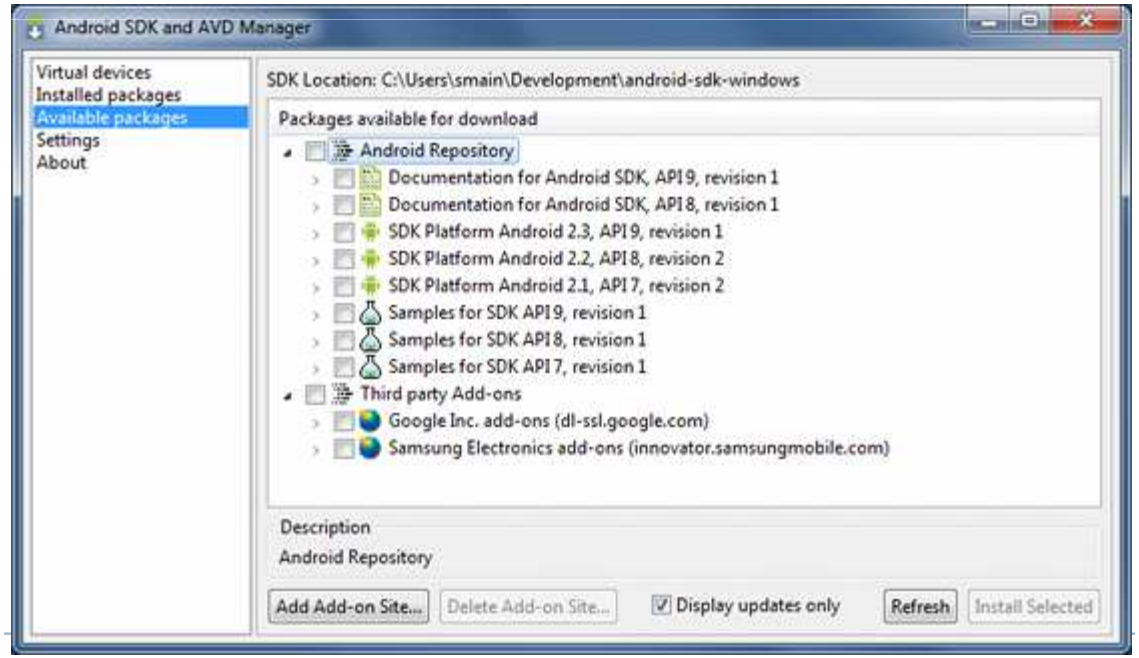
Virtual Device Manager

- Le SDK Android et le Virtual Device Manager sont utilisés pour créer et gérer les AVD(Android Virtual Devices) et les packages du SDK.

SDK Manager

- C’est un outil qui permet de créer et gérer les appareils virtuels.

- Il permet aussi de voir la version du SDK installée et d’installer de nouvelles.

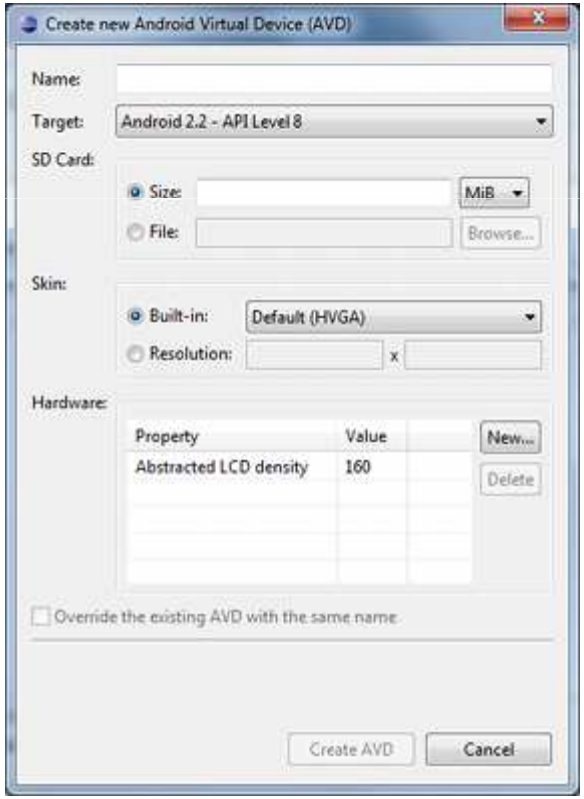


Périphérique virtuel (AVD)

- Android Virtual device (AVD) permet de définir les carectéristiques matérielles du périphérique virtuel.
- Exemple vous pouvez définir si l'appareil a un caméra, si elle utilise un clavier.
- Permet de définir la version de la plate-forme Android qui sera exécutée sur le périphérique virtuel.
- Ainsi tester vos applications sur plusieurs matériels sans acheter les téléphones correspondants.

Périphérique virtuel (AVD)

- Vous pouvez spécifier l'émulateur que vous souhaitez utiliser avec le périphérique virtuel.
- Chaque appareil configuré a un nom, une version Android, une capacite de carte SD et une résolution d'écran.



Machine virtuelle Dlavik

- Dalvik c’est la machine virtuelle utilsée par Android pour les mobiles.
- Elle permet d’exécuter des applications spécifiques sur n’importe quelle smartphone Android.
- Elle garantie qu’un appareil puisse efficacement exécutées plusieurs applications Android.
- Elle utilise le noyau Linux sous-jacent de l’appareil pour gérer l’interaction de bas-niveau avec le matériel.

Emulateur

- L’emulateur d’Android est un outil de test et de débogage d’application Android.
- Il fournit une connexion réseau complète, une simulation d’envoi et de réception d’appels et de SMS.
- C’est une implémentation de la machine virtuelle Dalvik, faisant de celle-ci une plateforme exécutant les applications Android comme le fait n’importe quel téléphone Android (Téléphone réel).
- Intégré au Plugin ADT Eclipse, l’Emulateur est lancé automatiquement avec l’AVD séléctionné lors d’une exécution ou débogage.

Emulateur

- Hors Eclipse l'émulateur peut s'exécuter via Telnet et le contrôler depuis une console : **emulator –avd <avd_name>.**

- Remarque : il faut créer un ou plusieurs AVD que vous associez à l'émulateur.

- NB : l'émulateur n'implémente pas toutes les caractérisques des matériels mobiles supportées par Android.

Emulateur

- Exemple d’émulateur.

Dalvik Debug Monitoring Service (DDMS)

- L'émulateur permet de voir le comportement et la ressemblance de l'application.

- Mais c'est le DDMS qui permet de voir ce qui ce passe en profondeur.

- Le DDMS est un outil puissant de débogage avec lequel on peut :
  1. Interroger les processus actifs;
  2. Examiner la stack et le heap;
  3. Surveiller et mettre en pause les threads actifs et explorer le système de fichier de n'importe quel matériel Android connecté;

Dalvik Debug Monitoring Service (DDMS)

- Le DDMS est intégré à Eclipse via le plugin ADT, Il est disponible aussi dans une perspective.

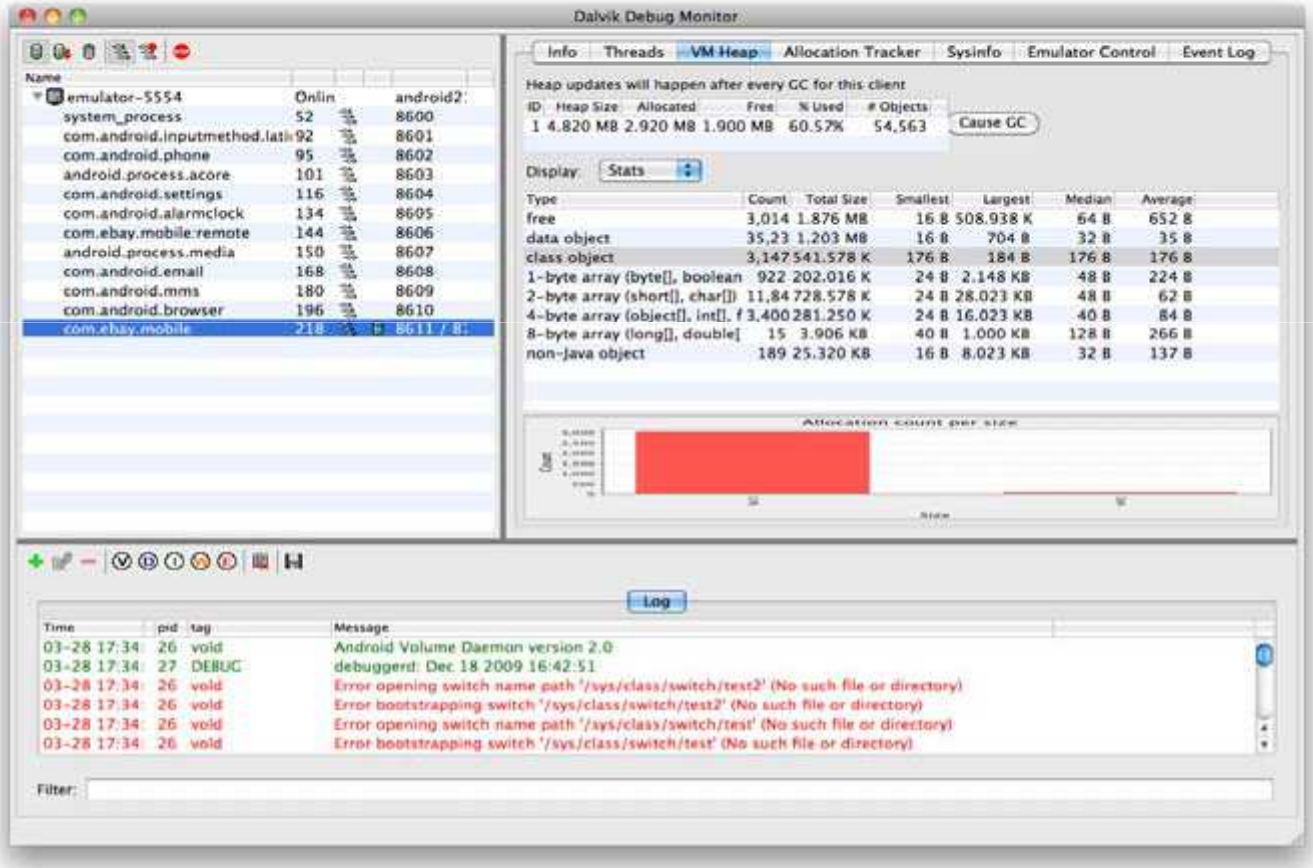
- La perspective sous Eclipse fournit un accés simplifié aux captures d'écran de l'émulateur et aux journaux générés par LogCat.

- Il est possible de lancer le DDMS en ligne de commande pour se connecter à tout appareil ou émulateur actif.

- Sur le répertoire tools du SDK avec un terminal, entrée la commande: **ddms** ou(**./ddms** sur Mac/Linux)

Dalvik Debug Monitoring Service (DDMS)

- Le perspective ddms sous eclipse.



Android Assert Packaging Tool (aapt)

- C’est un outil Android pour packager les applications dans un fichier zip d’extension .apk.
- Il peut être utilisé directement pour afficher, créer et mettre à jour des .zip, .jar et des archives APK mais aussi compiler des ressources.
- Il est inclu dans le répertoire tools du SDK.
- Pour l’exécuter aller dans le répertoire tools du SDK et lancer : **aapt.exe** sur Windows ou **./aapt** sur (Linux ou Mac).

Création package .apk

- La création du package peut se faire en utilisant Assistant Export ADT avec le plugin ADT.
- L'assistant d'exportation permet de compiler l'application, générer une clé privé(si nécessaire), et signer l'APK.
- Le package APK peut être créer aussi manuellement en utilisant **ANT.**
- Il existe deux mode de compilation:
  1. Debug mode (**ant debug**)
  2. Release mode (**ant release**)

Android Debug Bridge(ADB)

- ADB est à la fois un client et un service qui permet de se connecter à un émulateur Android ou appareil.

- Android Debug Bridge est composé de trois parties:
  1. Un démon exécuté par l'émulateur;
  2. Un service exécuté par la machine de développement;
  3. Des applications clientes(comme le DDMS) qui communiquent avec le démon via le service;

Android Debug Bridge(ADB)

- Le client ADB peut se lancer par la commande **adb**.
- Le plugin ADT automatise et simplifie les intéraction avec l'ADB.
- Le client vérifie toujours s'il existe un processus serveur en cours d'exécution.
- Le client démarre le processus serveur s'il n'existe pas.
- Le serveur se lie au port local TCP 5037 et écoute les commandes du client ADB.

Android Debug Bridge(ADB)

- Les clients ADB utilisent le port 5037 pour communiquer avec le serveur.

- Le serveur établie alors les connexions en localisant les instances d'émulateurs.

- Chaque instance de périphérique/émulateur nécessite deux ports.
  1. Un port pair pour la connexion console;
  2. Un port impair pour la connexion ADB;

- Une fois les connexions établissent, utilisez les commandes de l'ADB ou le plugin ADT pour contrôler les instances.

SQLite

- SQLite est un système de bases de données relationnelles (SGBD).

- Ses caractéristique principales sont:
  1. Open source;
  2. Compatible avec les standards;
  3. Léger;
  4. Mono tiers.

- Il a été implémenté sous forme d'une bibliothèque C compacte incluse dans Android.

- Chaque application crée sa propre base de données.

Traceview

- Traceview est une visionneuse graphique pour les logs enregistrés par les applications.
- Il aide à déboguer l'application et le profil de ses performances.
- Pour utiliser traceview, créer un fichier de trace en utilisant la classe de débogage dans le code.
- Exemple :
  1. Debug.startMethodTracing("nomFichier") pour démarrer le suivi;
  2. Debug.stopMethodTracing() pour arrêter le suivi;
- Ces methodes peuvent être utilisées respectivement dans le onCreate, onDestroy de l'activiy.

Traceview

- A l’appel de la méthode **stardMethodtracing**, le système commence l’enregistrement des données tracées jusqu’à l’appel de la méthode **stopMethodTracing**.

- Ensuite le système enregistre les données en mémoire dans le fichier de sortie.

- Si la taille maximale du tempon est atteinte avant **stopMethodtracing**, le système arrête le traçage et envoie une notification à la console.

- Pour visualiser le fichier de trace, exécutez la commande: **traceview** nomFichier.

mksdcard

- L’outil mksdcard permet de créer une image disque FAT32.
- L’image disque peut être charger dans l’émulateur pour simuler la présence d’une carte SD dans l’appareil.
- Pour son utilisation :

  mksdcard [-l nom] <taille> [k|M] <fichier>

- Une fois le disque image crée, la commande emulator – sdcard <fichier> le charge dans l’émulateur.

Dx

- C’est un outil qui permet de créer du bytecode Android à partir de fichier .class.
- Il convertit les fichiers et/ou répertoires en exécutable Dalvik sous le format .dex, pour le faire fonctionner dans l’environnement Android.
- Il peut également restaurer le fichier .class en format lisible(.java).

activityCreator

- activityCreator est un programme fournit par le SDK Android.
- Il peut être utiliser pour créer un nouveau projet ou un projet existant.
- Pour les machines Windows le SDK fournit un script batch appelé **activityCreator.bat**.
- Pour les machines Linux, Mac le SDK fournit un script Python appelé **activitycreator.py.**
- Le programme est utilisé de la même manière quel que soit le système d'exploitation.

activityCreator

- Pour exécuter activityCreator et créer un projet :
  1. Aller dans le répertoire tools du SDK et créer un nouveau répertoire pour vos fichiers;
  2. Exécuter activityCreator. Dans la commande il faut spécifier un nom de classe comme argument;
  3. Si c'est un nouveau projet la classe représente le nom d'une classe stub que le script va créer;
  4. Si c'est un projet existant la classe représente une classe activity dans le package;
- Commande :activityCreator.exe – out monProjet (mettez le nom complet avec le package).

layoutOpt

- C’est un outil de ligne de commande, il est disponible dans le SDK tools à partir de la révision 3.
- Il permet d’optimiser les layouts de l’application.
- Il peut être lancer sur les fichiers de configurations ou les répertoires de ressources pour vérifier les performances.
- Pour exécuter l’outil ouvrir un terminal et lancer : layoutopt <ressources>
- Loutil charge ensuite les fichiers XML spécifiés et analyse leur structures. Il affiche des informations s’il existe des problèmes.

Compilation

- La compilation sous Android peut être automatiser en utilisant le plugin ADT.
- Android SDK utilise aussi ANT pour automatiser la compilation.
- Android 2.2 Froyo intégre le compilateur JIT(Just in Time compiler) stable.

Conception

Contraintes de développement pour mobiles

- Le développement mobile ne manque pas de contraintes.

- Les contraintes se dressent devant les développeurs d'applications mobiles contribuant à complexifier significativement leur travail.

- Les principaux obstacles au développement sont:
  1. Les contraintes de test;
  2. Le faible nombre d'API disponibles;
  3. Les limitations de mémoire;
  4. La migration de plate-forme;
  5. Des problèmes de performances;

Développer pour Android

- Il existe plusieurs facteurs à prendre en compte lors d'un développement mobile.

- Tenir compte du matériel :
  1. Une puissance processeur faible;
  2. Une RAM limitée;
  3. Petits écrans avec faibles résolutions;
  4. Des coûts élevés de transferts de données;
  5. Des connexions réseau moins fiables;
- Efficacité : optimiser votre code afin qu'il soit rapide et réactif.

Developper pour Android

- Les considérations de conception précédentes sont importantes dans tout développement mobile.

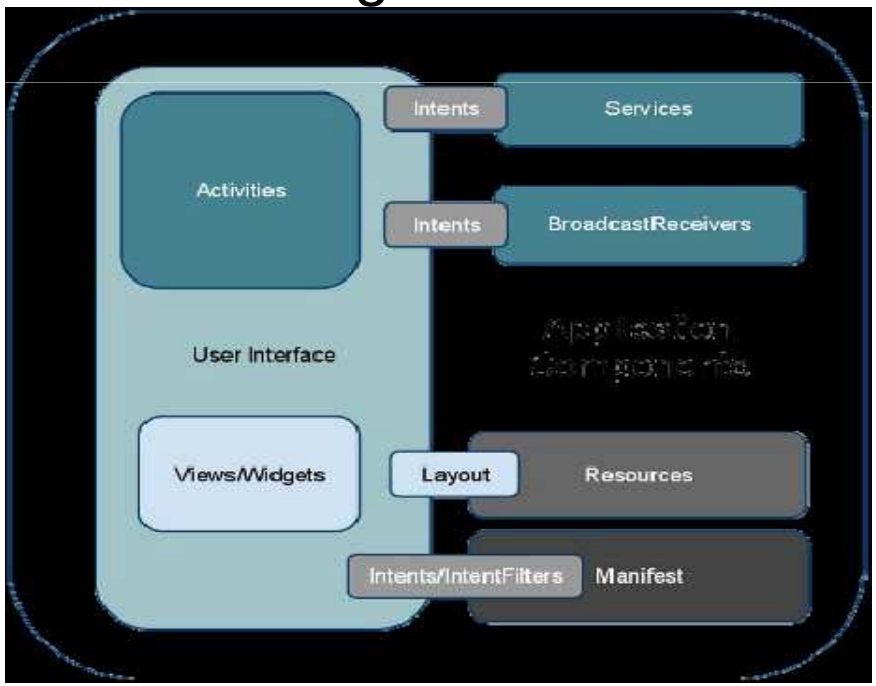
- Android impose quelques considérations particulières en plus de ces régles générales.

- Les applications Android doivent être conçues pour :
  1. La performance;
  2. La réactivité;
  3. La sécurité;
  4. La transparence;

Composants d'une application Android

- Les applications Android sont constitués de composants à couplage.

- Les composants sont liés par un manifeste d'application qui décrit chacun d'eux et comment ils interagissent.



Composants d'une application Android

- Les composants suivants sont en quelque sorte les briques élémentaires de vos applications:
  1. **Activities** : qui est la couche de presentation de votre application;
  2. **Services** : les composants qui tournent en arrière plan;
  3. **Content providers** : Sources de données partageables;
  4. **Intens** : Framework de communication interapplications.
  5. **Broadcast receivers** : Consommateurs des messages diffusés par les intents.
  6. **Widgets** : Composant d'application visuels;
  7. **Notifications** :Framework de notifications au utilisateurs;

Les 4 types d'application Android

- La plupart des applications Android appartiennent à l'une des catégories suivantes :
  1. **Application de premier plan**: c'est une application qui est utilisable uniquement lorsqu'elle est visible et mise en suspens lorqu'elle ne l'est pas;
  2. **Application d'arrière plan**: c'est une application à interaction limitée et qui, reste la plupart du temps cachée;
  3. **Intermittente**: c'est une application qui présente une certaine interactivité mais effectue l'essentiel de sa tâche en arrière plan. Ces applications notifient l'utilisateur lorsque cela est nécessaire;
  4. **Widget**: ces applications ne sont représentées que sous la forme d'un widget de l'écran d'accueil;

Fichier et éditeur Manifest.xml

- Chaque projet Android inclut un fichier aux format XML nommé manifeste (AndroidManifest.xml).
- Le manifeste permet de définir la structure et les métadonnées de votre application, ses composants et ses prérequis.
- Il contient des balises pour chacun des composants qui constituent votre application (Activities, Services, Content Providers et Broadcast Receivers)
- Ces balises seront définit au chapitre Fichier Manifest.

Fichier et éditeur Manifest.xml

- Le manifeste fournit aussi des attributs permettant de spécifier les métadonnées d'une application (Exemple son icône, son thème).

- Il fournit également des noeuds pour les paramètres de sécurité, les tests unitaires et la définition des prérequis matériels et de plateforme.

- Le fichier manifeste (AndroidManifest.xml) se trouve à la racine du projet.

Fichier et éditeur Manifest.xml

- Strucutre du fichier manifeste:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
```

Fichier et éditeur Manifest.xml

- Strucutre du fichier manifeste (suite):

```
<activity-alias>
        <intent-filter> . . . </intent-filter>
        <meta-data />
    </activity-alias>
    <service>
        <intent-filter> . . . </intent-filter>
        <meta-data/>
    </service>
    <receiver>
        <intent-filter> . . . </intent-filter>
        <meta-data />
    </receiver>
    <provider>
        <grant-uri-permission />
        <meta-data />
    </provider>
    <uses-library />
  </application>
</manifest>
```

Fichier et éditeur Manifest.xml

- Le plugin ADT eclipse comprend un éditeur visuel qui évite de manipuler directement le XML.

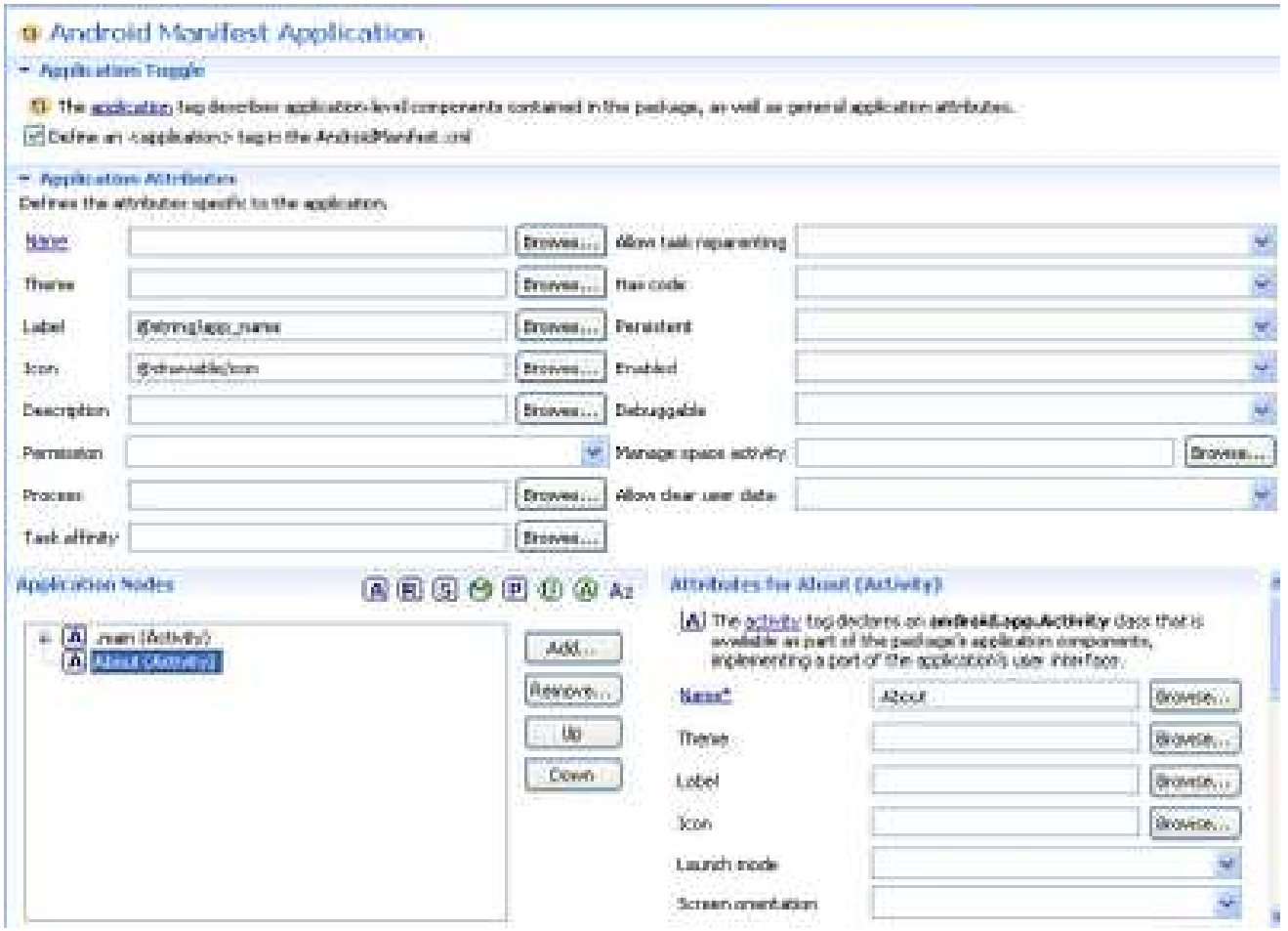
- Pour y accéder : clic droit sur le fichier AndroidManifest.xml situé dans votre dossier de projet et séléctionnez Open Wiht ..>Android Manifest Editor.

- Cet éditeur permet ainsi de spécifier les attribut d'une application(icône, label, theme) dans le panneau Application Attributes.

- Le panneau Application Nodes permet de gérer les composants de l'application(cf :Ecran ci-dessous).

Fichier et éditeur Manifest.xml

- Editeur de manifeste sous éclipse:



Cycle de vie application Android

- Les applications Android ont un contrôle limité sur leur propre cycle de vie. Leurs composants doivent être à l'écoute des changements d'état de l'application et réagir en conséquence pour une fin intempestive.

- Par défaut, chaque application Android est exécutée dans son propre processus, exécutant une instance distincte de la machine Dalvik.

- Celle-ci se charge de la mémoire et la gestion de processus.

- Les processus sont tués sans avertissements pour libérer des ressources pour d'autres applications.

Priorités des applications

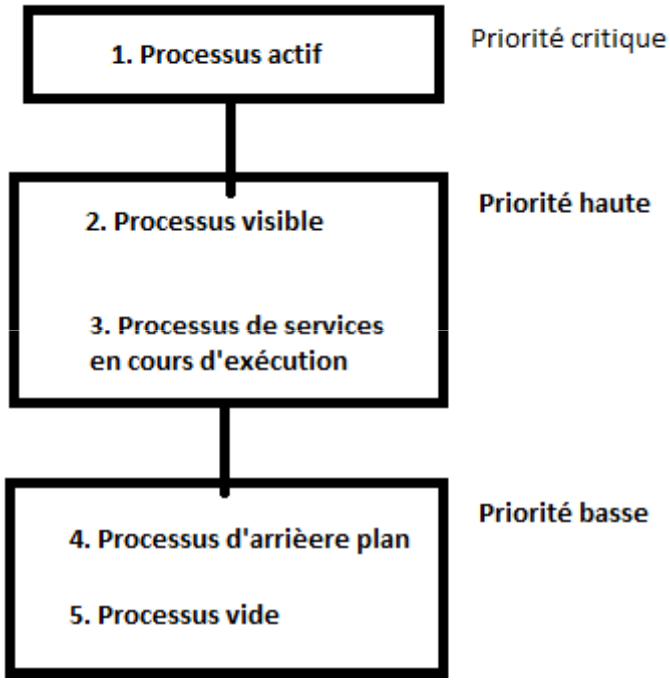
- L’ordre dans lequel les processus sont tués pour libérer les ressources est déterminé par la priorité de leur application.

- La prorité d’une application est celle du composant de plus haute priorité.

- Les priorités peuvent se classer en 3 types:
    1. Priorité critique: le processus est actif;
    2. Priorité haute: le processus est visible ou processus de service en cours d’exécution;
    3. Priorité basse: processus d’arrière plan ou processus vide;

Priorités des applications

- L’arbre des priorités utilisé pour déterminer l’ordre de fin des applications.



Etat des processus

- **Processus actifs** : Sont les processus au premier plan, les composants qui interagissent avec l'utilisateur. Android les gardent à l'état actif en récupérant des ressources.

- **Processus visible** : Son les processus qui hébergent des Activities "visible" mais à l'état inactif. Ils sont tués aux cas extrêms pour permettre aux pressesus actifs decontinuer à s'exécuter.

- **Processus de services en cours d'exécution** : Son les processus hébergeant des services ayant été démarrés. Ils s'exécutent sans interface utilisateur et en continue. Ils sont tués que si des ressources sont necessaires aux processus actifs ou visibles.

Etat des processus

- **Processus d'arrière-plan**: Sont les processus hébergeant des Activities non visible et n'ayant aucun service en cours d'exécution. Ils sont tués pour récupérer des ressources au profit des processus de premier plan.

- **Processus vide** :Pour des raisons de performance, Android garde des applications en mémoire une fois celles-ci terminées. Ce cache réduit le temps de relance des ces applications. Cependant ces processus sont tués si necessaire.

Ressources XML / Java

- Android supporte plusieurs types de ressources (Voir Chapitre des **Ressources**).
- A la compilation d'une application Android, l'outil **AAPT** ou le plugin **ADT** génère la clase R qui contient les Ids des ressources crées.
- Il existe 2 manières d'accéder aux ressources :
    1. Par code : en tuilisant la classe statique R. Exemple **R.string.hello** (où string est le type de ressource et hello le nom de la ressource) ;
    2. Par XML : en utilisant la syntaxe XML qui correspond aussi à l'ID de ressource définie dans laclasse R. Exemple @string/hello (où string est le type de ressource et hello le nom de la ressource);

Classe Application

- La classe Application d'Android permet:
  1. Maintenir l'état d'une application;
  2. Transférer des objets entre les composants d'une application;
  3. Gérer et maintenir les ressources utilisées par plusieurs composants d'une application;

- Enregistrez la classe Application dans le manifeste aprés l'avoir implémentée. Ceci donne accès à ses méthodes et variables.

- L'implémentation de la classe est instanciée lorsque le processus de l'application est créé (C'est un singleton).

Classe Application

- Exemple d'implémentation et d'enregistrement dans le manifeste:

```
Public  class MyApplication extends Application {
   private static Myapplication appSingleton;
   //renvoie l'instance de l'application
   public static MyApplication getInstance() {
        return appSingleton;
   }
   @override
   public final void onCreate() {
        super.onCreate();
        appSingleton=this;
   }
}
```

Classe Application

- Exemple d'implémentation et d'enregistrement dans le manifeste: (suite)

```
<application
   android:icon="@drawable/icon"
   android:name="MyApplication">
   [...Noeuds du manifeste....]
</application>
```

- Ainsi vous pouvez créer de nouvelles variables d'état et de ressources globales pour y accéder depuis les composants de l'application.

Exemple :
```
MyObject value = MyApplication.getInstance().getGlobalStateValue();
MyApplication.getInstance().setGlobalStateValue( myObectValue);
```

Classe Application

- La classe application fournit également des gestionnaires d'évenements pour le démarrage et la fin d'une application, de la méoire disponible et de changement de configuration:
    1. onCreate : pour le démarrage de l'application;
    2. onTerminate : pour la fin de l'application;
    3. onLowMemory : fournit aux applications pour libérer de la mémoire lorsque le système manque de resource;
    4. onConfigurationChanged : qui permet de gérer les changement de configuration au niveau de l'application;

Activity : la classe

- La classe Activity permet de créer des écrans en utilisant des Views pour interagir avec l'utilisateur.

- Chauqe Activity représente un écran (Form).

- Exemple :

```
public class MyActivity extends Activity {
   //Appelée lorsque l'activité est crée pour la 1ere fois
   @override
   public void onCreate(Bundle saveInstanceState){
        super.onCreate(saveInstanceState);
   }
}
```

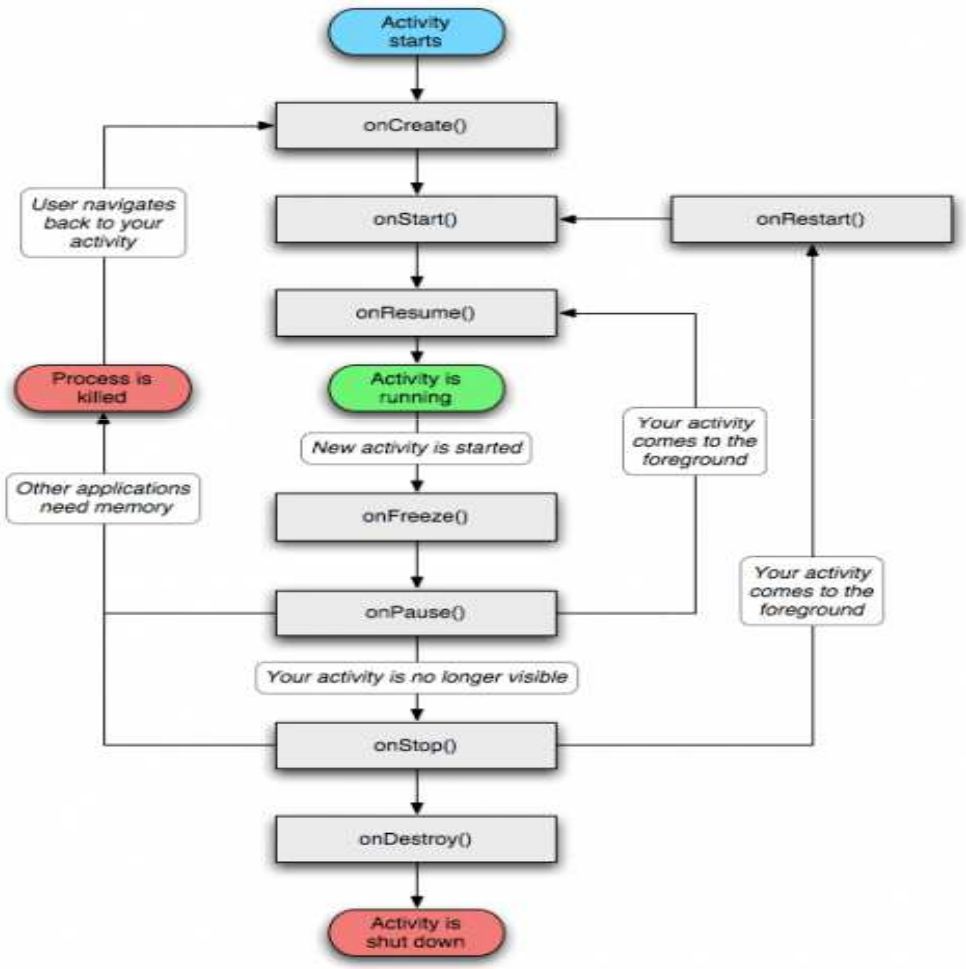
Activity : le cycle de vie

- Une bonne compréhension du cycle de vie de l'Activity permet de garantir que votre application satistfait vos utilisateur et gère correctement ses ressources.

- L'état de chaque Activity est déterminé par sa position dans la pile des Activities:
    1. Active : Quand l'Activity est au sommet de la pile;
    2. En pause : Si l'Activity est visible sans avoir le focus;
    3. Stoppée : Lorsque l'Activity n'est plus visible;
    4. Inactive : Quand elle est tuée ou qu'elle n'est pas démarrée;

Activity : le cycle de vie

- Schéma d'un cycle de vie d'une Activity:



Interface utilisateur

Fondamentaux

- Pour bien créer des interfaces utilisateurs, il faut :
    1. Avoir une bonne conception d'interface utilisateur;
    2. Tenir compte de l'interaction homme-machine;
    3. Considérer aussi la facilité d'utilisation;

- Android introduit une nouvelle terminologie pour des méthaphores de programmation familières(Activity, View, View Group) que nous allons voir dans la section suivante.

Activity, View, View Group

- **Activity** = écran Android (GUI). Il représente les fenêtres ou les écrans affichés. Une Activity contient des views.

- **View** = composant graphique (Widget). C'est la classe de base, tous les contrôles d'interface sont dérivés de View.

- **View Group** : c'est une extension de la classe View. Il contient plusieurs Views enfants. Son extension permet de :
    1. Créer des contrôles composites interconnectées;
    2. Fournir les gestionnaires de layouts pour disposer les contrôles dans vos Activities;

Views personnalisées

- Android permet de créer des views personnalisés, non disponible sur les composants graphiques de bases.

- L'extension des classes View ou SurfaceView permet de créer des composants views(widgets).

- La classe View fournit un objet Canvas(surface sur laquelle nous dessinons notre vue) ainsi qu'une serie de méthodes de dessin et de classes Paint.

- La classe SurfaceView fournit un objet Surface qui supporte le dessin depuis un thread en arrière-plan et utilise openGL pour la 3D. Elle est excélente pour les contrôles gourmands en graphisme.

Views personnalisées

- Pour créer une vue personnalisée vous pouvez redéfinir les méthodes :
    1. onMeasure : qui calculera la hauteur et la largeur que la vue occupera;
    2. onDraw : qui supportera le Canvas;
- Exemple:

```
@Override
protected void onMeasure(int width,int height){
        [..... Définir la hauteur et la largeur...]
}
@Override
protected void onDraw(Canvas canvas){
        [..... Dessiner votre interface visuelle ...]
}
```

Widgets Android

- Android fournit une boîte à outils de Views(widgets) standard qui facilite la création des interfaces;

- Parmis ses contrôles on peut citer :
  1. TextView : pour les labels texte en lecture standard en lecture seule;
  2. EditText : boite de saisie modifiable;
  3. ListView : View Group créant et gérant une liste verticale de Views et les affichant sous forme de lignes dans une liste;
  4. Button : bouton standard;
  5. CheckBox : bouton à deux états représentant une case à cocher;

Layouts : Linearlayout, FrameLayout

- Les layouts sont des extensions de la classe **ViewGroup**, utilisés pour placer des contrôles enfants dans une interface utilisateur.

- Le SDK Android inclu quelques layouts simples.

- **Linearlayout** : aligne les views verticalement ou horizontalment. Il contrôle la taille relaive des views.

- **FrameLayout** : c'est le plus simple, il supperpose les views enfants au coin supérieur gauche.

Layouts : RelativeLayout, TableLayout, Gallery

- **RelativeLayout**: c'est le plus flexible des layouts natifs. Il permet dedéfinir les positions chaque view par rapport aux autres et aux limites de l'écran.

- **TableLayout**: permet de placer les views sur une lgrille de lignes et de colonnes.

- **Gallery** : affiche une ligne unique d'objets dans une liste déroulante horizontale.

Layouts : Utilisation

- Les layouts peuvent être implémenter en utilisant du XML ou du code.

- Exemple d'utilisation **LinearLayout** avec XML:

```
<?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:text="Entrez votre texte ici"
    />
    <EditText
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
    android:text="Le texte s\'affiche là !"
    />
</LinearLayout>
```

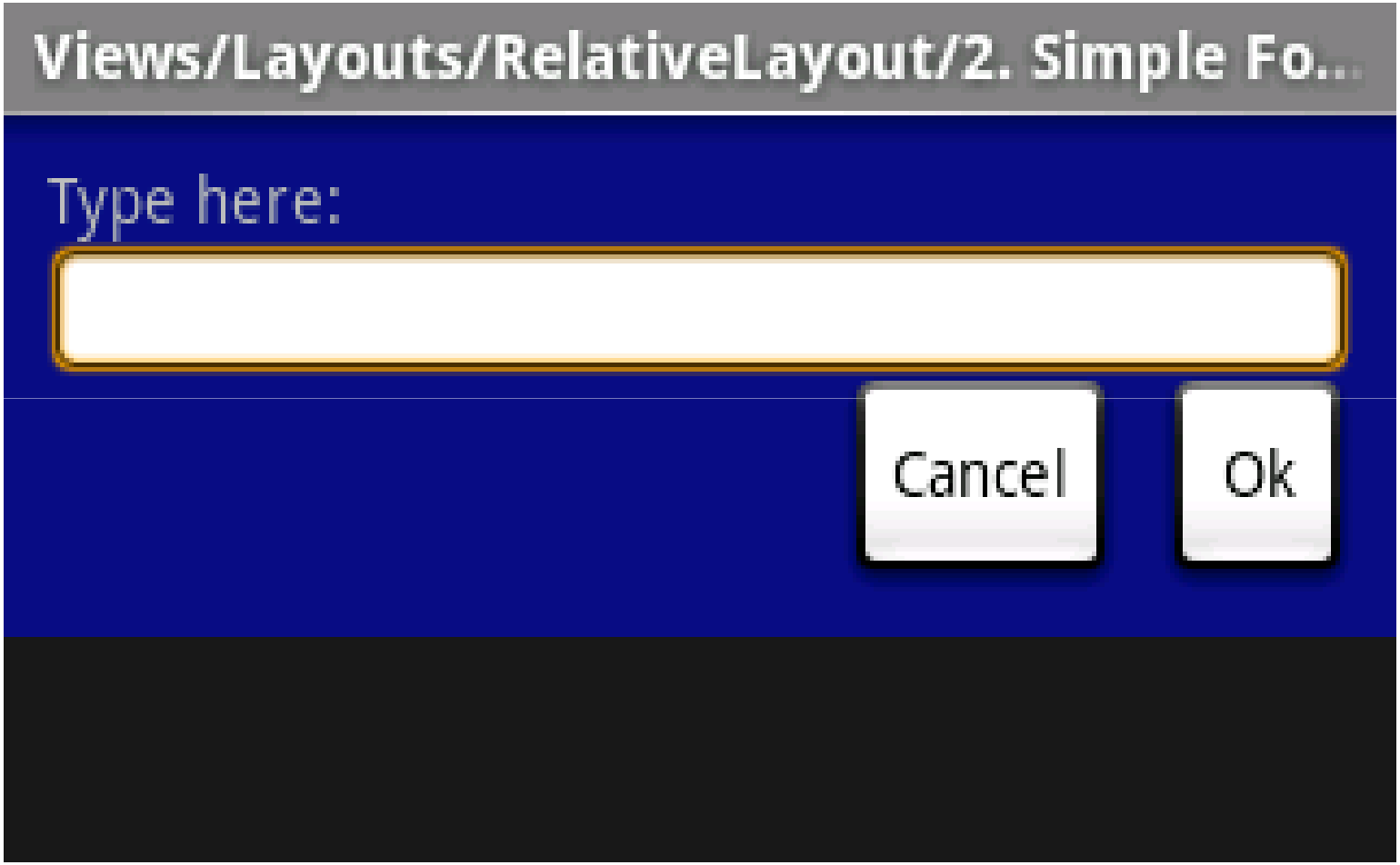
Layouts : Utilisation

- Emple d'utilisation **LinearLayout** avec du code:

```
LinearLayout ll = new LinearLayout(this);
ll.setOrientation(LinearLayout.VERTICAL);
TextView myTextView = new TextView(this);
EditText myEditText = new EditText(this);
myTextView.setText("Entrez votre texte ici");
myEditText.setText("le texte s'affiche là !");
int lHt = LinearLayout.LayoutParam.FILL_PARENT;
int lW = LinearLayout.LayoutParam.WRAP_CONTENT;
ll.addView(myTextView, new LinearLayout.LayoutParams(lH, lW);
ll.addView(myEditTet, new LinearLayout.LayoutParams(lH, lW);
setContentView(ll);
```

- Remarque : La notion de fill_parent a été remplacée par wrap_parent dans la version 2.2 d'Android.

Layouts : Exemple écran



Ressources drawable

- Android contient plusieurs types de ressources Drawable.

- Les ressources sont sont stockées dans le dossier res/drawable.

- Les ressouces drawable peuvent être définies et manipulées par du XML ou du code.

Ressources drawable

- Exemple d'utilisation avec une ressource image.

```
<ImageView
   android:layout_height="wrap_content"
   android:layout_width="wrap_content"
   android:src="@drawable/myimage" />

Resources res = getResources();
Drawable drawable = res.getDrawable(R.drawable.myimage);
```

- Où le référencement :
  - En Java : R.drawable.myimage
  - En XML : @drawable/myimage

Résolution

- 2009-2010 constitue une explosion du nombre d'appareils Android(large gamme de résolutions d'écran).
- Les interfaces développées doivent tenir compte de cette large gamme de résolution.
- Les applications Android doivent pouvoir s'exécuter sur une variété de matériels indépendamment.
- Le framework Android fournit des techniques qui permettent d'optimiser l'interface utilisateur pour fonctionner dans différentes tailles d'écran.

Résolution

- Qualificateurs de ressources : Structurer votre projet en créant des répertoires pour stocker vos ressources externes pour différentes configurations matérielles.

- Exemple de qualificateurs :
    1. res/layout-small-long/ : Layout pour de petits et longs écrans;
    2. res/layout-large/ : Layout pour de grands écrans;
    3. res/drawable-hdpi/ : Drawables pour des écrans à haute densité;

Résolution

- Possibilité d'optimiser vos interfaces en utilisant le fichier manifest.

- Exemple :

```
<supports-screens
   android:smallScreens="false"
    android:normalScreens="true"
    android:largeScreens="true"
    android:anyDensity="true"
/>
```

- L'attribut anyDensity contrôle comment l'application sera mise à l'echelle quand elle s'affiche sur des appareils de densités diverses.