INFO0406 - Programmation sur Smartphone

Sommaire

- Structure d'un projet Android
- Boites à outils pour Android

Rappels

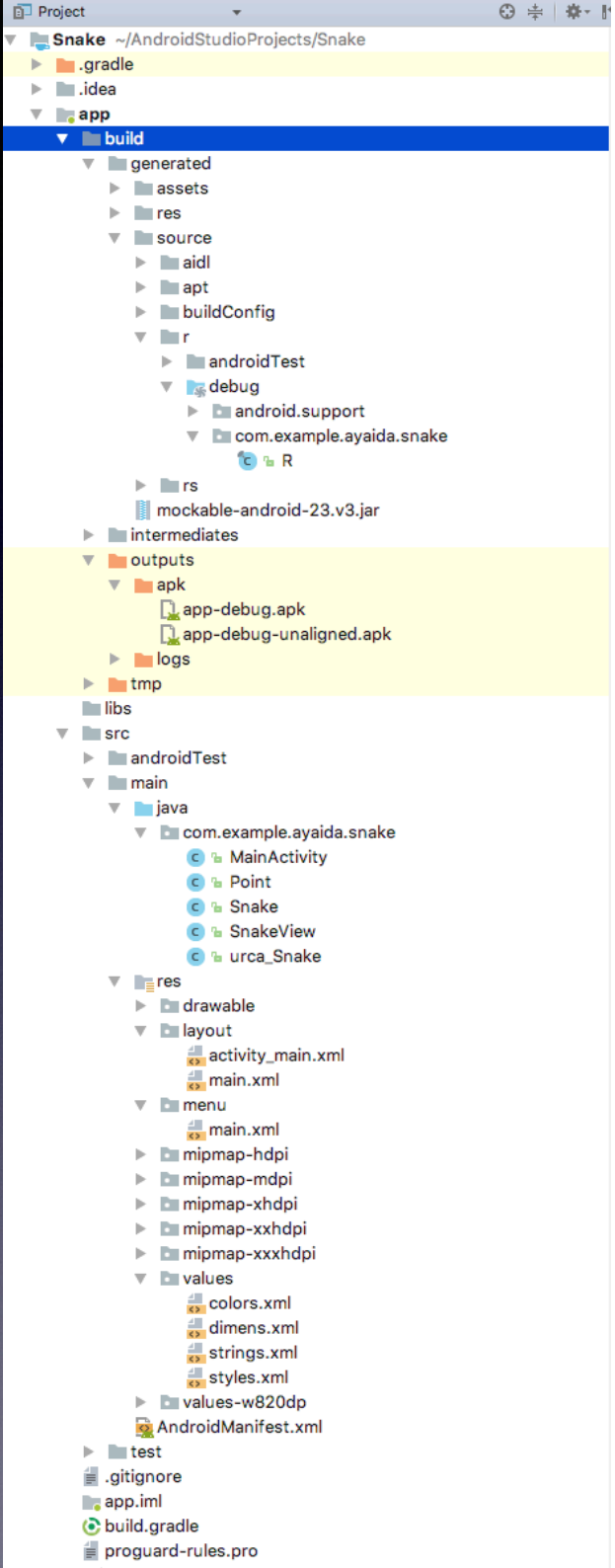
- Interface d'une applications Android
- Layout d'applications
  - Agent de placement
  - Fichiers XML
  - Exemples : *LinearLayout, RelativeLayout, TableLayout, ScrollView*
- Composants graphiques d'une application
  - ***TextView***
  - ***Button***
  - ***ImageView*** & ***ImageButton***

Structure d'un projet Android

Faculté des Sciences Reims

Structure d'un projet Android

- Fichier AndroidManifest.xml
  - Décrit l'application et les composants
- Répertoires
  - build/ : contient les fichiers compilés
    - output/apk/ : contient l'application compilée
    - genrated/r/ : contient R.java fichier auto généré décrivant les ressources
  - src/ : contient les fichiers sources
    - main/java/ : vos sources .java organisés en package
    - main/res/ : contient les ressources organisées par spécialisation (drawable, layout, menu, binaires, chaînes)



Gérer les ressources de son application

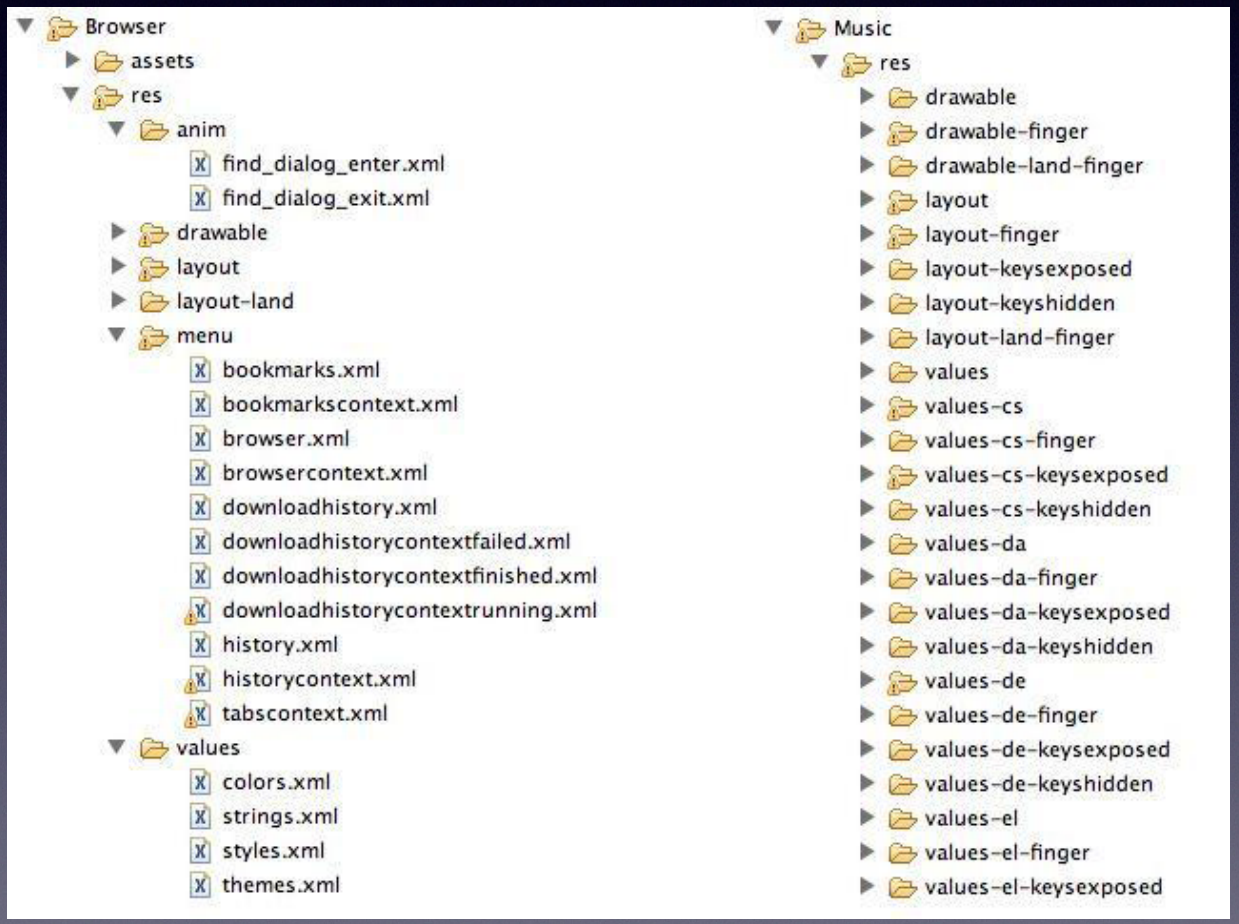
- Les ressources sont des données statiques stockées en dehors du code java.
  - Dans un projet Android, les ressources sont stockées sous le répertoire res.
- Avec Android vous pouvez gérer simplement des ensembles de ressources par configuration (écran, langue, densité, interaction, …)
- Les différents types de ressources:
  - layout: contient les xml décrivant la mise en page
  - anim: contient les animations courtes de l'interface utilisateur
  - drawable: contient les images et icônes de l'application
  - values: contient les chaînes, les couleurs, les tableaux et les dimensions
  - xml: contient nos propres données au format xml
  - menu: contient la description des menus
  - mipmap : contient l'icône de l'application avec plusieurs résolutions

Les avantages du système Android

- L'accès simplifié aux ressources
  - On accède aux ressources grâce à une constante générée dans le R.java.
  - Ex: R.drawable.background pour l'image background.png.
- Gestion simple de la spécialisation des ressources
  - La spécialisation se base sur le nom des répertoires.
  - Le répertoire racine contient les ressources par défaut les branches puis les feuilles des ressources plus spécialisées.
  - Attention: toujours bien définir des ressources par défaut
- Localisation
  - Vous pourrez facilement traduire votre application car les ressources sont gérées dans un fichier externe.

Gestion de la spécialisation des ressources

- Les paramètres de spécialisation sont séparés par des tirets.
- Le système va toujours chercher la ressource la plus spécialisée



Externaliser la gestion des ressources

- Externaliser la gestion des ressources vous permettra de gérer simplement les configurations spécifiques des terminaux Android (Taille d'écran…).
- Pour externaliser la gestion des ressources, vous devez organiser les ressources de votre projet dans le répertoire res/ et ses sous répertoires.
- Pour tous les types de ressources vous pouvez définir:
  - Des ressources par défaut
    - Layout par défaut sauvé dans res/layout/
  - Des ressources alternatives
    - Layout en cas d'orientation paysage res/layout-land/
- Application automatique par le système de la bonne configuration

Figure 1. Two different devices, both using default resources.

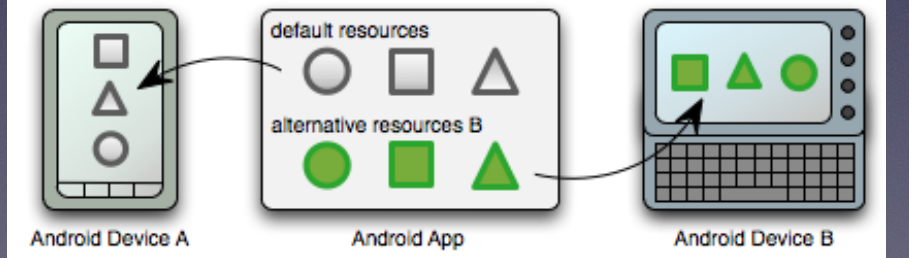


Figure 2. Two different devices, one using alternative resources.

Externaliser la gestion des ressources

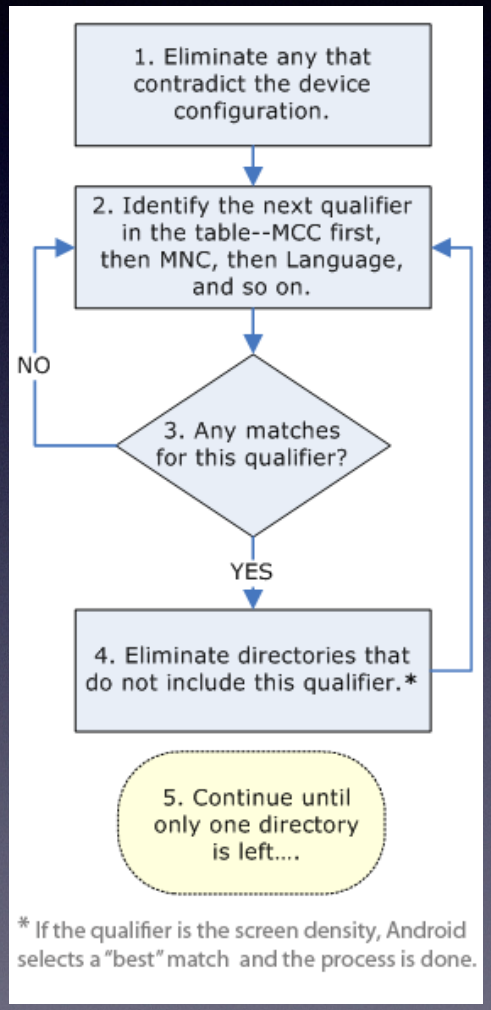
- Fournir des ressources
  - Produire les ressources pour chaque configuration cible
- Accéder aux ressources
  - Androïd propose un système d'accès aux ressources simplifié et centralisé
- Gérer les changements à l'exécution
  - Androïd gèrera le chargement des ressources durant le RunTime
  - Le seul point d'attention: définir des configurations par défaut
- Localisation
  - Simplement vous pourrez également localiser votre application
- Types de ressources :
  - Animation
  - Couleur
  - Images
  - Layout
  - Menu
  - Texte
  - Style, etc.

Ext

Répertoire	Type de ressource
anim/	Fichiers XML définissant les animations.
color/	Fichiers XML définissant des liste de couleurs.
drawable/	Fichiers images
layout/	Fichiers XML définissant les layouts qui définissent un agencement de l'interface
menu/	Fichiers XML qui définissent les menus de l'application (options, menu contextuel, sous-menu…)
raw/	Fichiers binaires
values/	Fichiers XML qui contiennent des valeurs simples, telles que les chaînes, entiers, tableaux …
xml/	Divers fichiers de configuration XML (configuration de recherche, …)

Produire votre gestion spécifique des ressources

- Produire vos alternatives selon la nomenclature Androïd
  - Ex : drawable-port-hdpi
    - Ressources graphiques en mode portrait pour les devices hdpi.
- Les alias
  - Pour ne pas dupliquer vos ressources
- !! Important !! Ressources par défaut
  - Toujours définir une ressource par défaut
    - Langue / Image / Mode d'affichage (port / land)…
- Fournir les meilleurs ressources
  - Itération du système pour déterminer le répertoire



Accéder aux ressources

- Accédez aux ressources par leur identifiant et leur type .
- Ex:
  - dans le code : R.string.bonjourstr
  - dans le xml : @string/bonjourstr
- Le fichier R.java (auto généré)

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.

package p8.demo.p8sokoban;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int block=0x7f020000;
        public static final int diamant=0x7f020001;
        public static final int icon=0x7f020002;
        public static final int perso=0x7f020003;
        public static final int vide=0x7f020004;
        public static final int zone_01=0x7f020005;
        public static final int zone_02=0x7f020006;
        public static final int zone_03=0x7f020007;
        public static final int zone_04=0x7f020008;
    }
    public static final class id {
        public static final int SokobanView=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Boites à outils pour Android

SDK Android

- Le kit de développement Android (SDK) fournit l'environnement de travail pour développer, tester et débogguer des applications Android.
- Dans le SDK on trouve :
  - Les API Android :
    - Ils sont le coeur du SDK.
    - Composés de bibliothèques d'API Android.
    - Ils donnent au développeur accès à la pile Android.
  - Des outils de développement :
    - Ils permettent de compiler et débogguer vos applications.
  - Le virtual Device Manager et l'Émulateur :
    - Il fournit un meilleur environnement de test.
- Des exemples de code et un support en ligne

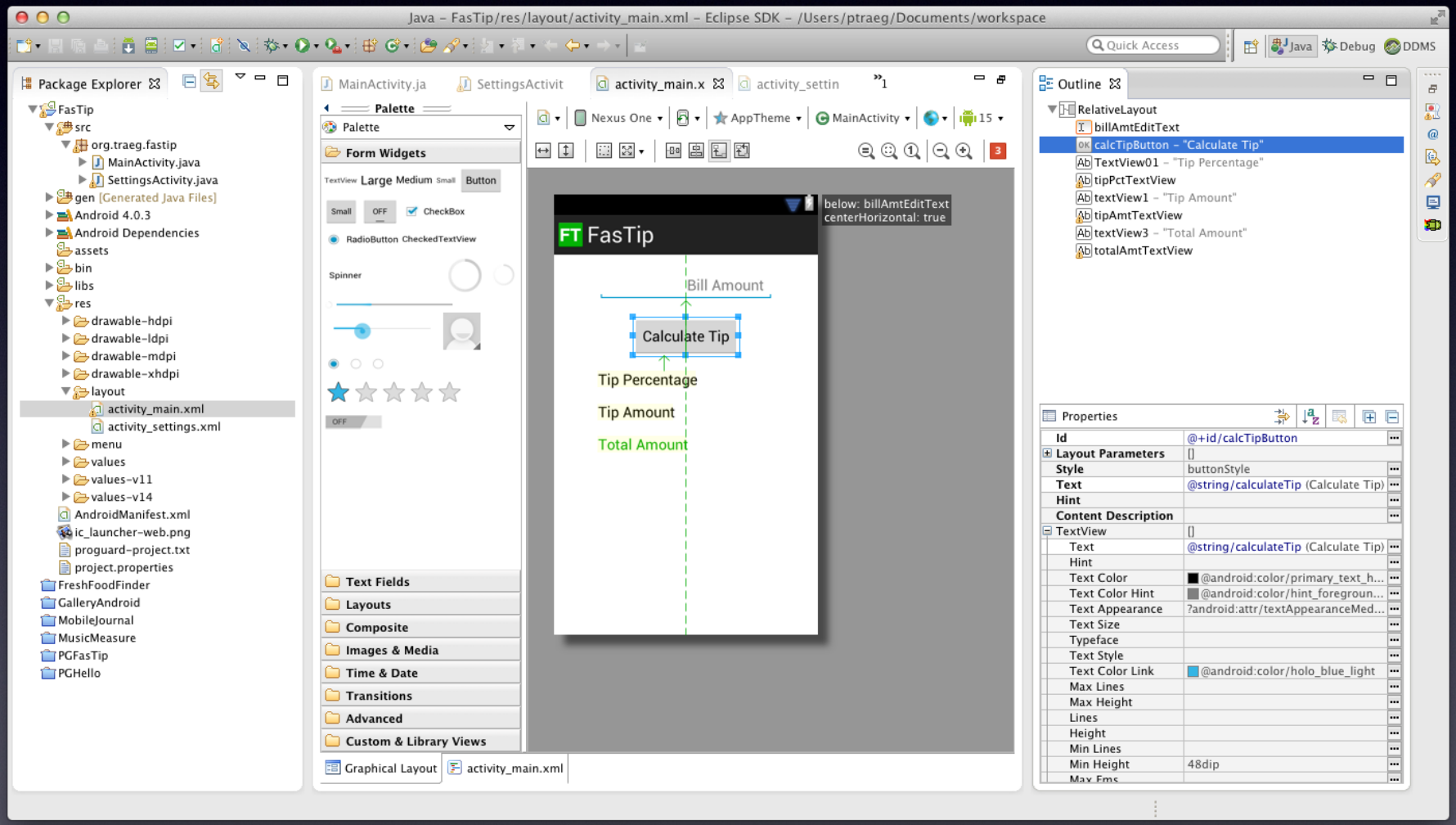
SDK Android

- Le SDK Android est disponible en téléchargement pour les plateformes Linux, 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 paramètres des applications.

Plugin Eclipse Android Development Tools (ADT)

- Le plugin eclipse ADT :
  - Il simplifie le développement Android.
  - Il intègre les outils de développement comme :
    - L'émulateur
    - 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.

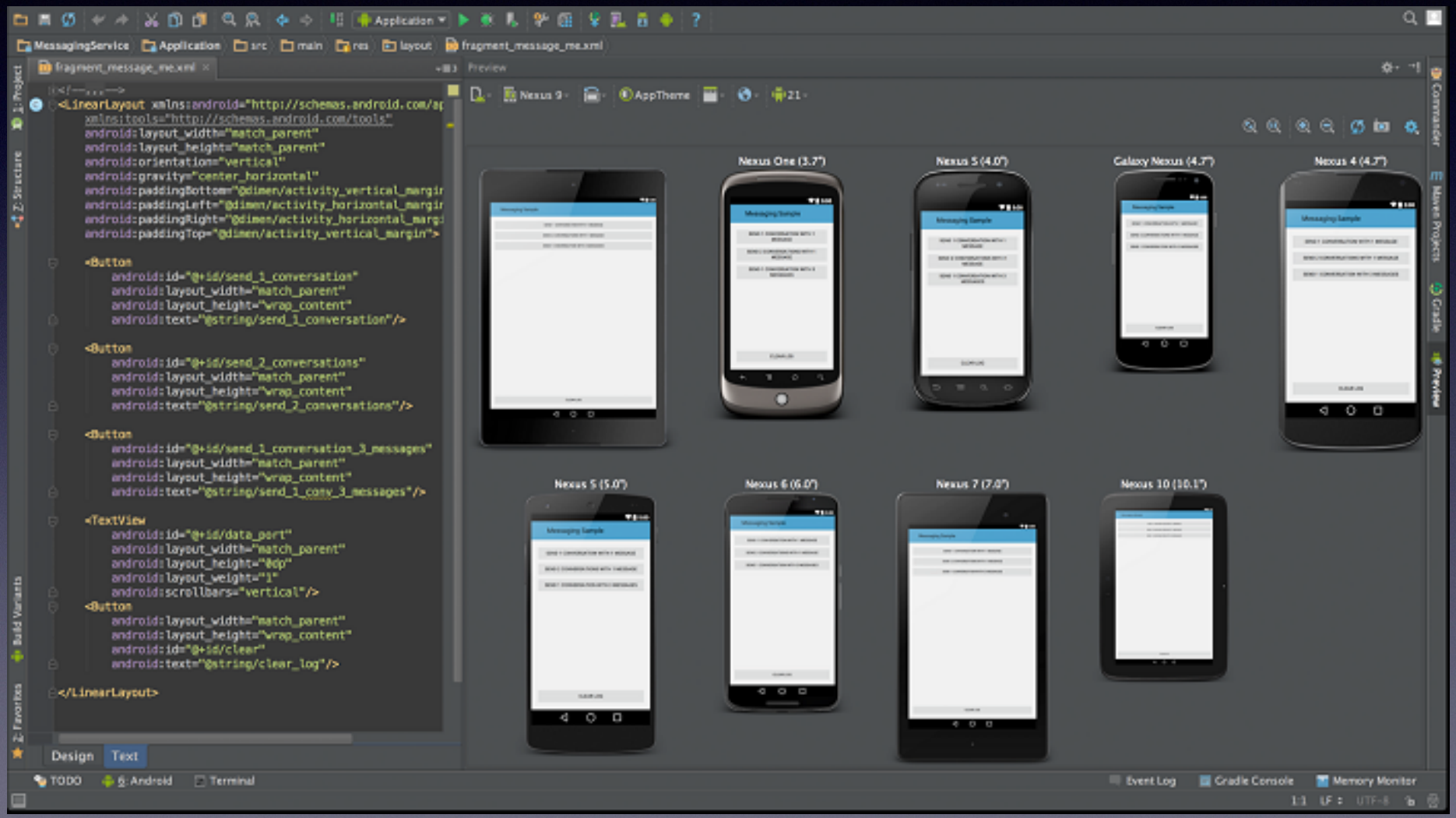
Plugin Eclipse ADT



Android Studio

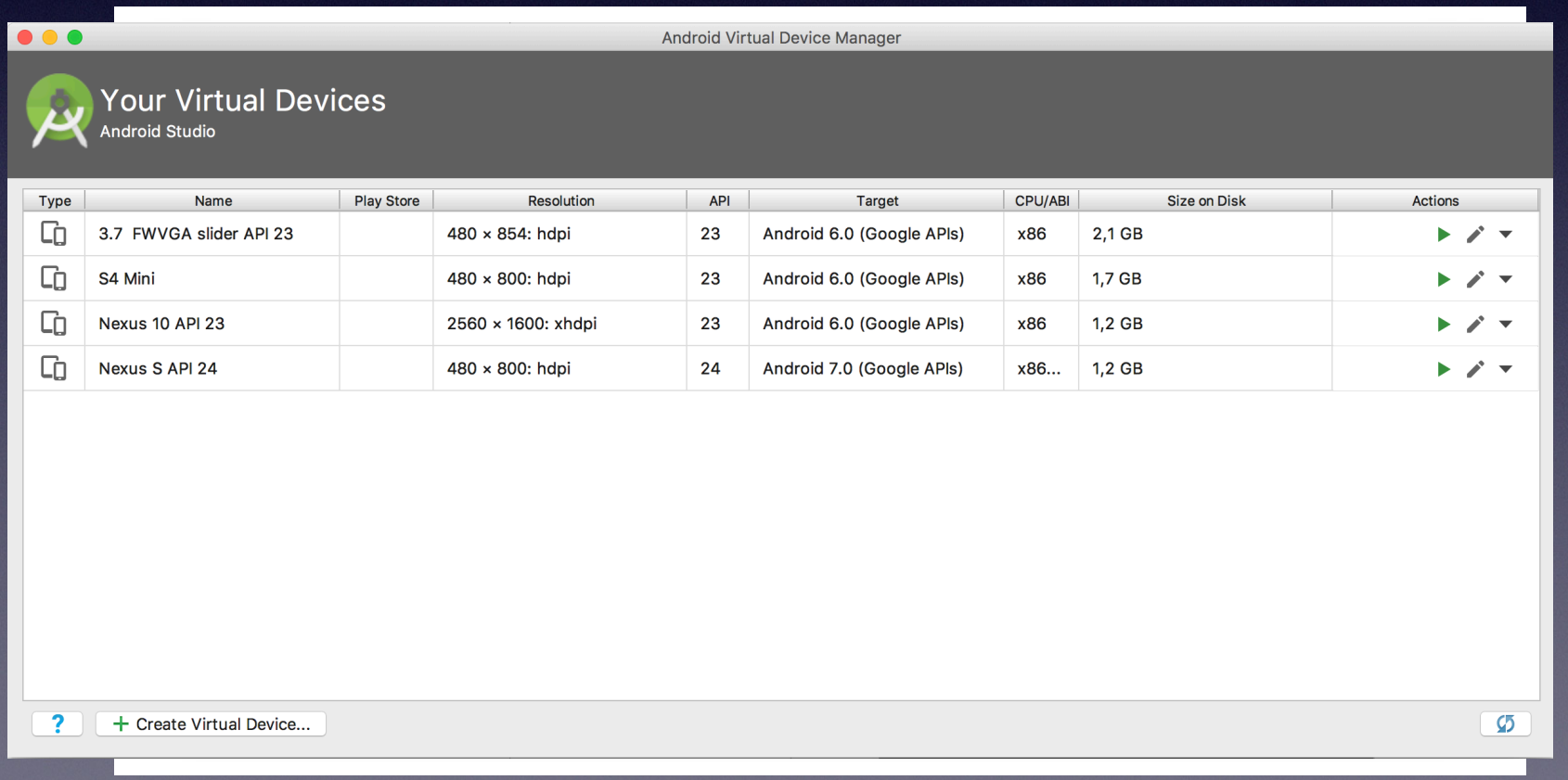
- Android Studio est annoncé le 15 mai 2013 lors du Google I/O
- Le 8 décembre 2014, Android Studio passe de version Bêta à version stable 1.0.
- Android Studio permet principalement d'éditer les fichiers Java et les fichiers de configuration d'une application Android.
- Il propose entre autres des outils pour gérer le développement d'applications multilingues
- Il permet de visualiser la mise en page des écrans sur des écrans de résolutions variées simultanément
- Il est mise à jour par Google
- Il remplace ADT qui n'est plus mise à jour

Android Studio



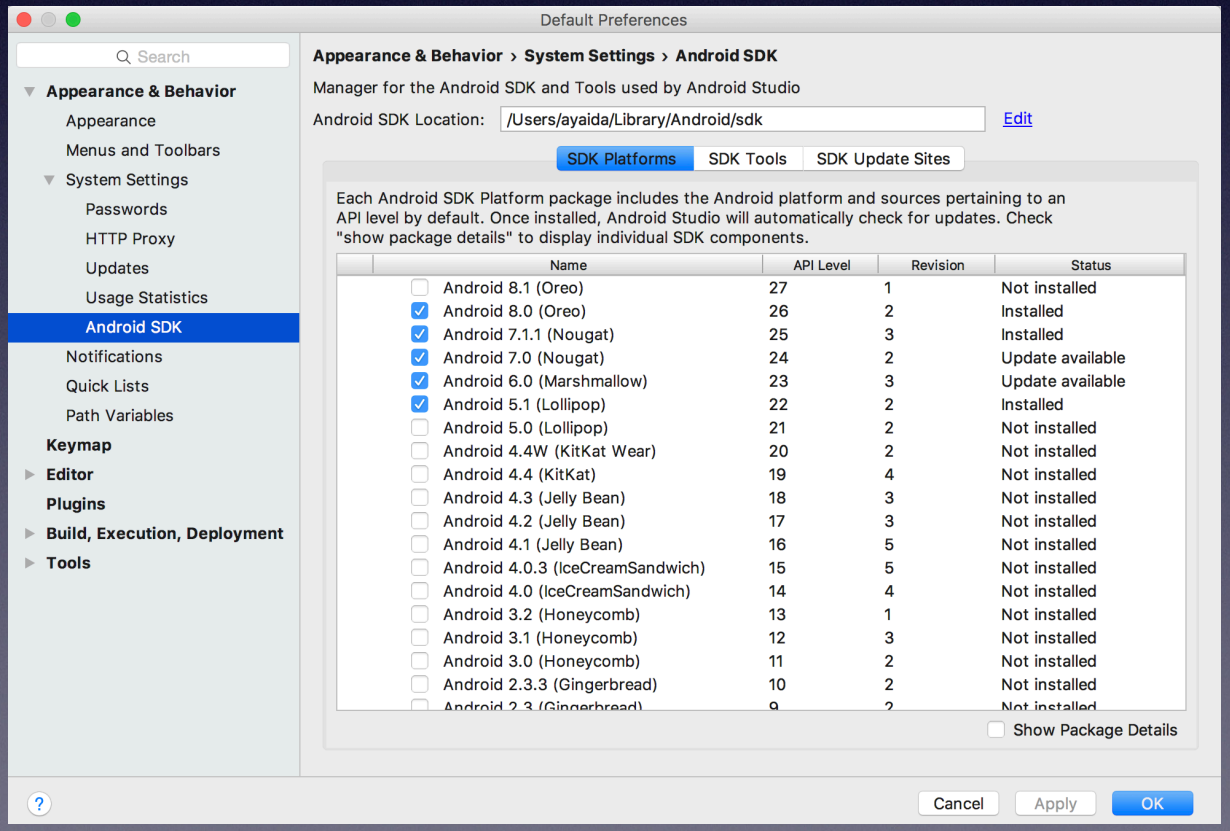
Virtual Device Manager (VDM)

- 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.

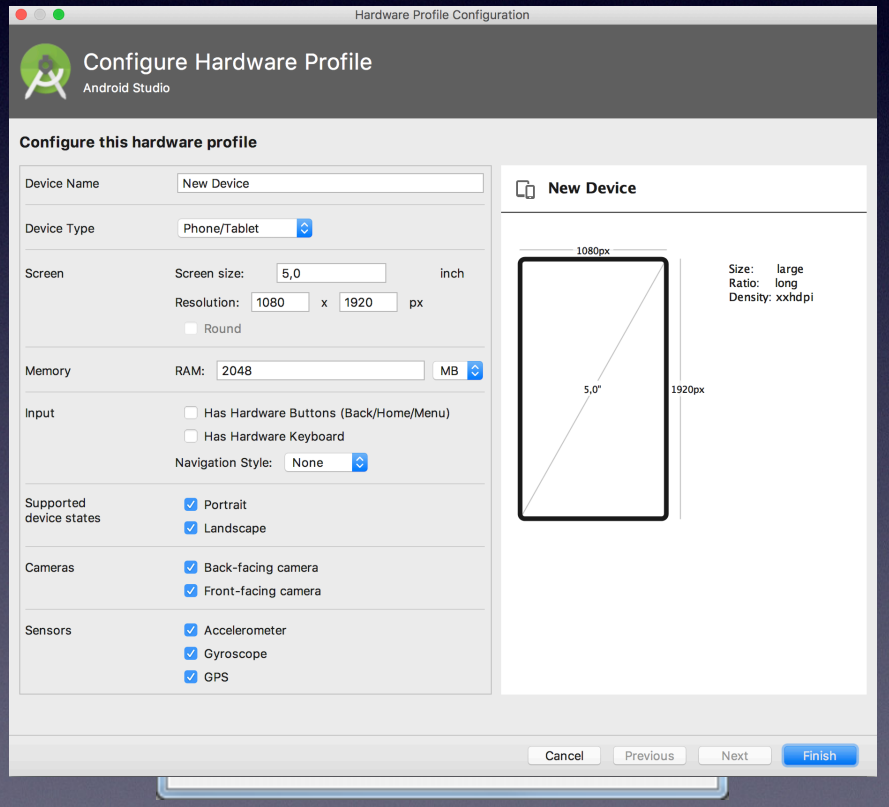


Périphérique virtuel (AVD)

- Android Virtual Device (AVD) permet de définir les caractéristiques matérielles du périphérique virtuel.
- Exemple : vous pouvez définir :
  - Si l'appareil possède une caméra,
  - S'il utilise un clavier,
  - etc.
- Permet de définir la version de la plate-forme Android qui sera exécutée sur le périphérique virtuel.
- Émuler différents périphériques
- Ainsi tester vos applications sur plusieurs matériels sans acheter les téléphones correspondants.

Périphérique virtuel (AVD)

- Name: Nom de votre choix (pas d'espace)
- Target: Version du SDK Android utilisée par l'émulateur
- SD Card: configuration de la mémoire externe
- Skins: résolution de l'émulateur (pré configuré ou personnalisée)
- Hardware: pour customiser l'émulateur (clavier, GPS, accéléromètre, densité,…)



Directement sur le mobile

- Passer le mobile en mode debug :
  - Avant Android 4.2 :
    - Bouton Home > Réglages > Applications > USB Debugging
  - A partir d'Android 4.2 : Menu caché
    - Bouton Home > Réglages > A propos du Téléphone > 7 fois sur Numéro du Build
    - Bouton Home > Réglages > Options Développeur > USB Debugging
- Le connecter en USB à la machine
- Passer le projet en choix manuel pour l'environnement d'execution
  - Clic droit sur le projet > Run/Debug Settings > éditer ou créer une configuration de lancement > target > sélectionner manuel
- Au prochain lancement de l'application depuis eclipse un « ***Popup*** » apparaîtra pour choisir l'environnement d'exécution.
- Sur le mobile il est possible de faire du pas à pas et de profiter de tout le confort de debug (log, point d'arrêt, thread, …).

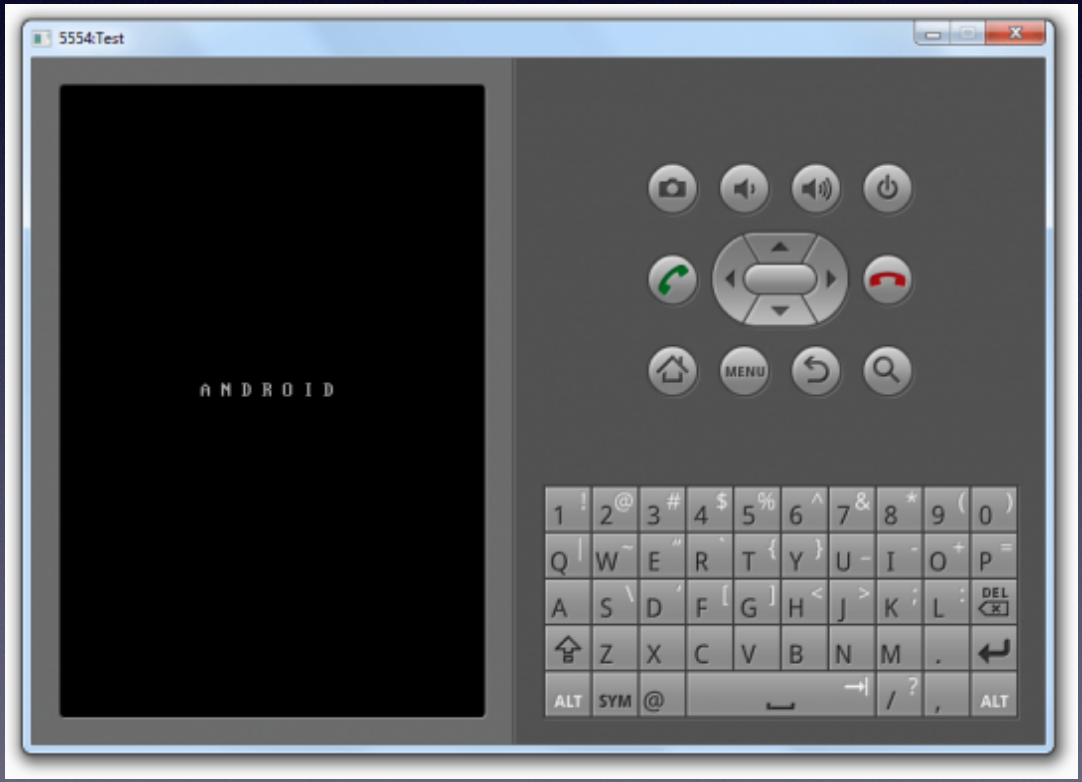
Machine virtuelle Dlavik

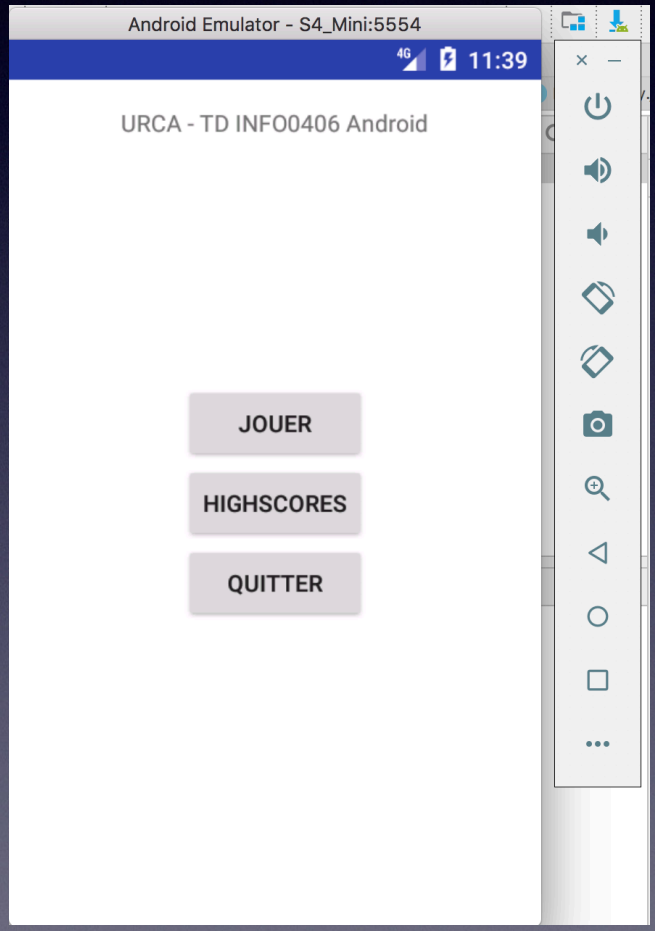
- Pour rappel « ***Dalvik*** » c'est la machine virtuelle utilisée par Android pour les mobiles.
- Elle permet d'exécuter des applications spécifiques sur n'importe quelle smartphone Android.
- Elle garantie que plusieurs applications Android puissent être efficacement exécutées sur un appareil Android.
- Elle utilise le noyau Linux sous-jacent de l'appareil pour gérer l'interaction de bas-niveau avec le matériel.

Émulateur Android

- L'émulateur d'Android est un outil de test et de déboggage 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).
- Il est intégré au Plugin ADT Eclipse et à Android Studio
- l'Émulateur est lancé automatiquement avec l'AVD sélectionné lors d'une exécution ou du déboggage.
- 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 caractéristiques des matériels mobiles supportées par Android.

Exemple d'Émulateur Android





Le Logcat

- Logcat vous permettra d'avoir toutes les informations sur l'exécution de l'émulateur.
- Logcat vous permet de définir plusieurs niveaux de logs qui seront filtrables :
  - Verbose
  - Debug
  - Info
  - Warning
  - Error
- Ajouter Logcat dans votre vue java:
  - Clic :Window>ShowView>Others>Logcat
- Appels :
  - Log.e(« groupe », « msg ») ou
  - Log.i(« groupe », « msg »)

Dalvik Debug Monitoring Service (DDMS)

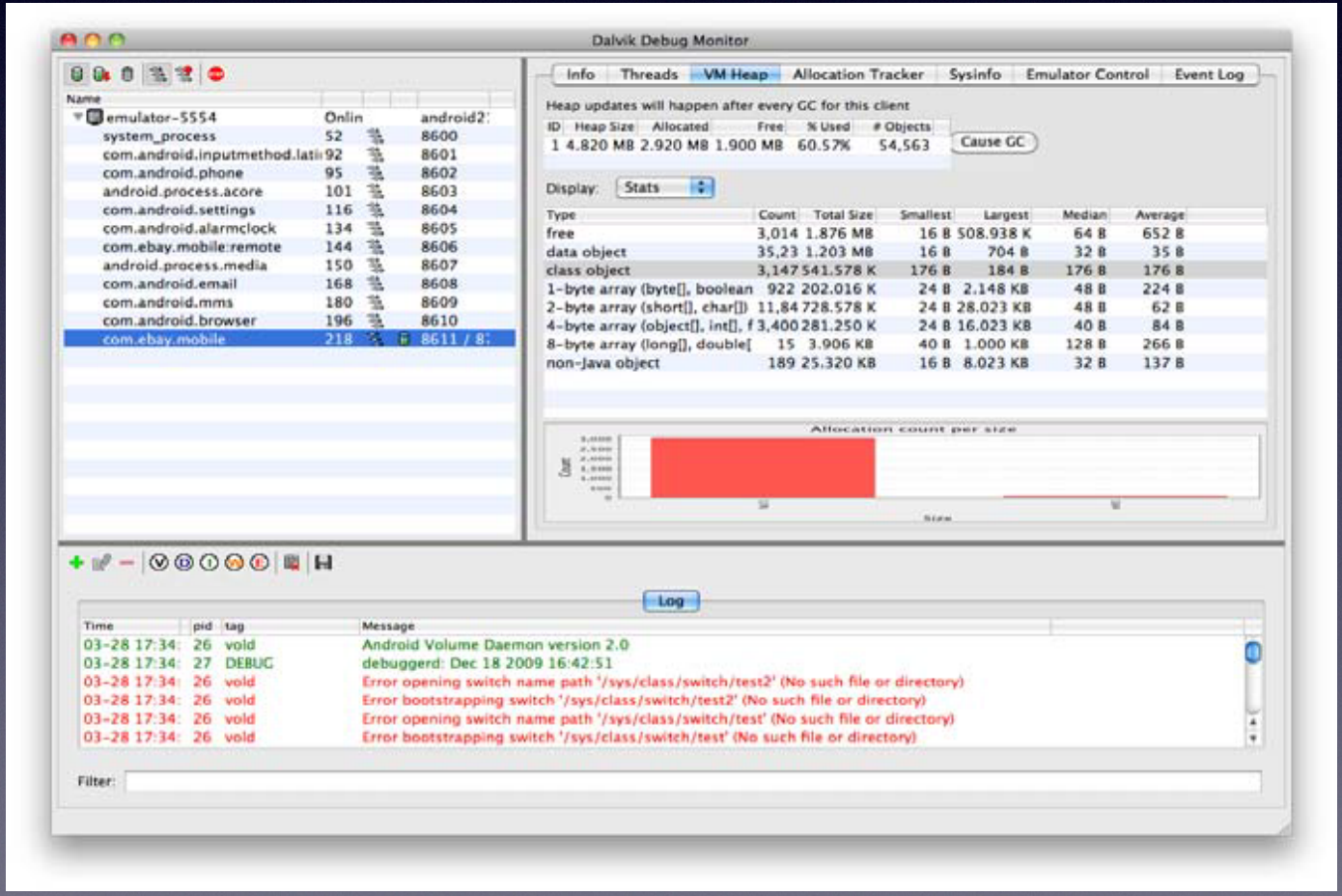
- L'émulateur permet de voir le comportement et la ressemblance de l'application.
- Mais c'est le DDMS qui permet de voir ce qui ce passe en profondeur.
- Le DDMS est un outil puissant de déboggage avec lequel on peut :
  - Interroger les processus actifs;
  - Examiner la stack;
  - Surveiller et mettre en pause les threads actifs et explorer le système de fichier de n'importe quel matériel Android connecté;
- Le DDMS est intégré à Eclipse via le plugin ADT, Il est disponible aussi dans une perspective.
- La perspective sous Eclipse fournit un accès 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)

- Comment ***DDMS*** interagit avec un débogueur?
- Sur Android, chaque application s'exécute dans son propre processus,
- chacun des processus s'exécute dans sa propre machine virtuelle (VM).
- Chaque VM expose un port unique avec lequel un débogueur peut attacher.
- Lorsque ***DDMS*** démarre, il se connecte à ***ADB***.
- Quand un appareil est connecté, un service de surveillance VM est créée entre la ***ADB*** et ***DDMS***, qui notifie ***DDMS*** quand un VM sur le périphérique est démarré ou arrêté.
- Une fois une machine virtuelle est en cours d'exécution, ***DDMS*** récupère l'identifiant du processus de la VM (***PID***), par l'intermédiaire du ***ADB***, et ouvre une connexion au débogueur de la machine virtuelle, via le démon ***ADB*** (***ADBD***) sur l'appareil.
- ***DDMS*** peut maintenant parler à la machine virtuelle en utilisant un protocole filaire personnalisé.

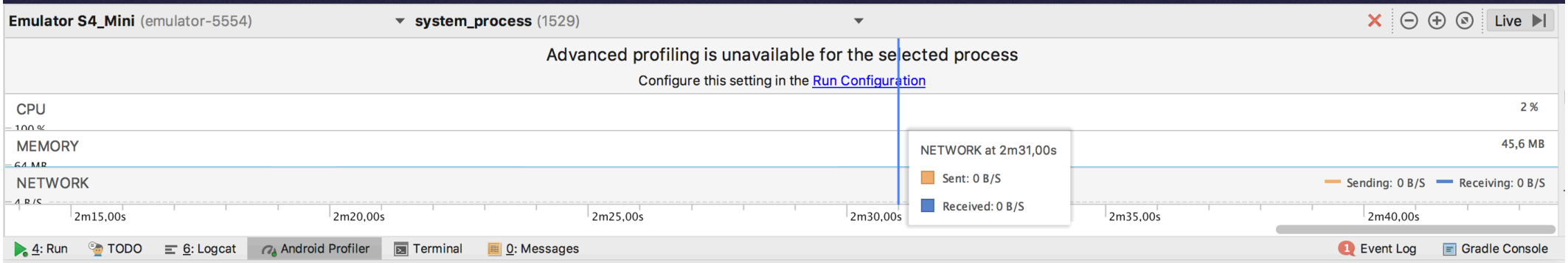
Dalvik Debug Monitoring Service (DDMS)

- Le perspective DDMS sous eclipse.



Android Profiler

- Le perspective Android Profiler successeur du DDMS sous Android Studio.



Android Asset 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 du Package .apk

- La création du package peut se faire en utilisant ***Assistant Export ADT*** avec le plugin ***ADT*** ou **Gradle Wrapper** sous **Android Studio**
- L'assistant d'exportation permet de compiler l'application, générer une clé privé(si nécessaire), et signer l'***APK***.
- Il existe deux mode de compilation:
  - Debug mode (ant debug)
  - 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 un appareil.
- ***Android Debug Bridge*** est composé de trois parties:
  - Un démon exécuté par l'émulateur;
  - Un service exécuté par la machine de développement;
  - Des applications clientes (comme le ***DDMS***) qui communiquent avec le démon via le service;
- Le client ***ADB*** peut se lancer par la commande « ***adb*** ».
- Le plugin ***ADT*** / ***Anroid Studio*** automatise et simplifie les interaction 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.
- Un port pair pour la connexion console;
- Un port impair pour la connexion ***ADB***;
- Une fois les connexions sont établies, utilisez les commandes de l'***ADB*** ou le plugin ADT pour contrôler les instances.

SQLite

- ***SQLite*** : une base de données évoluée et simplifiée
- ***SQLite*** est un système de bases de données relationnelles (***SGBD***).
- Android ne fournit aucune base de données. Si vous voulez utiliser ***SQLite***, vous devez créer votre base et la remplir.
- Ses principales caractéristique sont :
  - Open source;
  - Compatible avec les standards;
  - Léger;
- Elle a été implémentée 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 débogguer l'application et le profil de ses performances.
- Pour utiliser ***Traceview***, il faut créer un fichier de trace en utilisant la classe de déboggage dans le code.
- Exemple :
  - ***Debug.startMethodTracing***("nomFichier") pour démarrer le suivi;
  - ***Debug.stopMethodTracing***() pour arrêter le suivi;
  - Ces méthodes peuvent être utilisées respectivement dans le ***onCreate***, ***onDestroy*** de l'activité.

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 tampon 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***

Name	Incl %	Inclusive	Excl %	Exclusive	Calls+Rec
4 android/webkit/LoadListener.nativeFinished ()V	66.6%	17734.382	53.2%	14161.950	14+0
3 android/webkit/LoadListener.tearDown ()V	100.0%	17734.382			14/14
6 android/view/View.invalidate (IIII)V	19.8%	3516.410			2413/2853
57 android/webkit/BrowserFrame.startLoadingResource (ILjava	0.3%	44.636			3/15
53 java/util/HashMap.put (Ljava/lang/Object;Ljava/lang/Objec	0.0%	6.223			6/326
20 android/webkit/JWebCoreJavaBridge.setSharedTimer (J)V	0.0%	2.593			2/730
378 android/view/ViewGroup.requestLayout ()V	0.0%	1.139			2/54
315 java/util/HashMap.<init> (I)V	0.0%	0.879			3/41
629 android/webkit/BrowserFrame.loadCompleted ()V	0.0%	0.285			1/1
598 android/webkit/WebView.didFirstLayout ()V	0.0%	0.231			1/2
703 android/webkit/BrowserFrame.windowObjectCleared (I)V	0.0%	0.036			1/2
5 android/webkit/JWebCoreJavaBridge$TimerHandler.handleMessa	16.3%	4342.697	0.5%	132.018	730+0
6 android/view/View.invalidate (IIII)V	15.6%	4161.341	1.2%	319.164	2853+0
7 android/webkit/JWebCoreJavaBridge.access$300 (Landroid/webk	15.1%	4025.658	0.1%	26.727	729+0
8 android/webkit/JWebCoreJavaBridge.sharedTimerFired ()V	15.0%	3998.931	8.5%	2256.801	729+0
9 android/view/View.invalidate (Landroid/graphics/Rect;)V	13.8%	3671.342	0.9%	246.190	2853+0
10 android/view/ViewGroup.invalidateChild (Landroid/view/View;La	12.4%	3298.987	6.3%	1687.629	876+1148
11 android/event/EventLoop.processPendingEvents ()V	6.3%	1674.317	0.6%	151.201	12+0
12 android/view/ViewRoot.handleMessage (Landroid/os/Message;)	4.6%	1217.210	0.0%	1.992	35+0
13 android/view/ViewRoot.performTraversals ()V	4.5%	1209.815	0.0%	7.190	34+0
14 android/view/ViewRoot.draw (Z)V	4.1%	1096.832	0.0%	11.508	34+0
15 android/policy/PhoneWindow$DecorView.drawTraversal (Landrc	3.9%	1040.408	0.0%	2.218	34+0
16 android/widget/FrameLayout.drawTraversal (Landroid/graphics,	3.8%	1023.779	0.0%	3.129	34+48
17 android/view/View.drawTraversal (Landroid/graphics/Canvas;La	3.8%	1022.611	0.1%	19.213	34+154
18 android/view/ViewGroup.dispatchDrawTraversal (Landroid/grap	3.8%	1000.413	0.2%	42.609	34+130
19 android/view/ViewGroup.drawChild (Landroid/graphics/Canvas;	3.7%	983.346	0.2%	42.926	34+150
20 android/webkit/JWebCoreJavaBridge.setSharedTimer (J)V	3.5%	929.506	0.2%	57.241	730+0
21 android/webkit/WebView.nativeDrawRect (Landroid/graphics/Ca	3.5%	923.805	3.0%	807.952	15+0
22 android/net/http/QueuedRequest.start (Landroid/net/http/Queu	3.2%	847.172	0.0%	3.556	15+0
23 android/net/http/QueuedRequest$QREventHandler.endData ()V	3.1%	828.592	0.0%	1.619	15+0
24 android/net/http/QueuedRequest.setupRequest ()V	3.1%	819.888	0.0%	5.860	15+0
25 android/net/http/QueuedRequest.requestComplete ()V	3.1%	816.585	0.0%	1.506	15+0
26 android/webkit/CookieManager.getCookie (Landroid/content/Cc	2.7%	722.837	0.0%	8.081	15+0
27 android/webkit/LoadListener.commitLoad ()V	2.6%	688.168	0.1%	17.708	58+0
28 android/webkit/LoadListener.nativeAddData ([BI)V	2.3%	621.864	1.2%	306.817	57+0
29 android/graphics/Rect.offset (II)V	2.2%	573.985	2.2%	573.985	17210+0

Find:

mksdcard

- L'outil ***mksdcard*** permet de créer une image disque ***FAT32***.
- L'image disque peut être chargée 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 créée, la commande :
  - ***emulator – sdcard <fichier>***
  - Permet de le charge dans l'émulateur.

Dx

- Rappel : 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 :
  - Aller dans le répertoire tools du ***SDK*** et créer un nouveau répertoire pour vos fichiers;
  - Exécuter ***activityCreator***. Dans la commande il faut spécifier un nom de classe comme argument;
  - Si c'est un nouveau projet, la classe représente le nom d'une classe que le script va créer;
  - 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.
- Celui-ci est remplacé par « Créer Nouveau Projet » du Plugin ***ADT*** d'***Eclipse***

layoutOpt

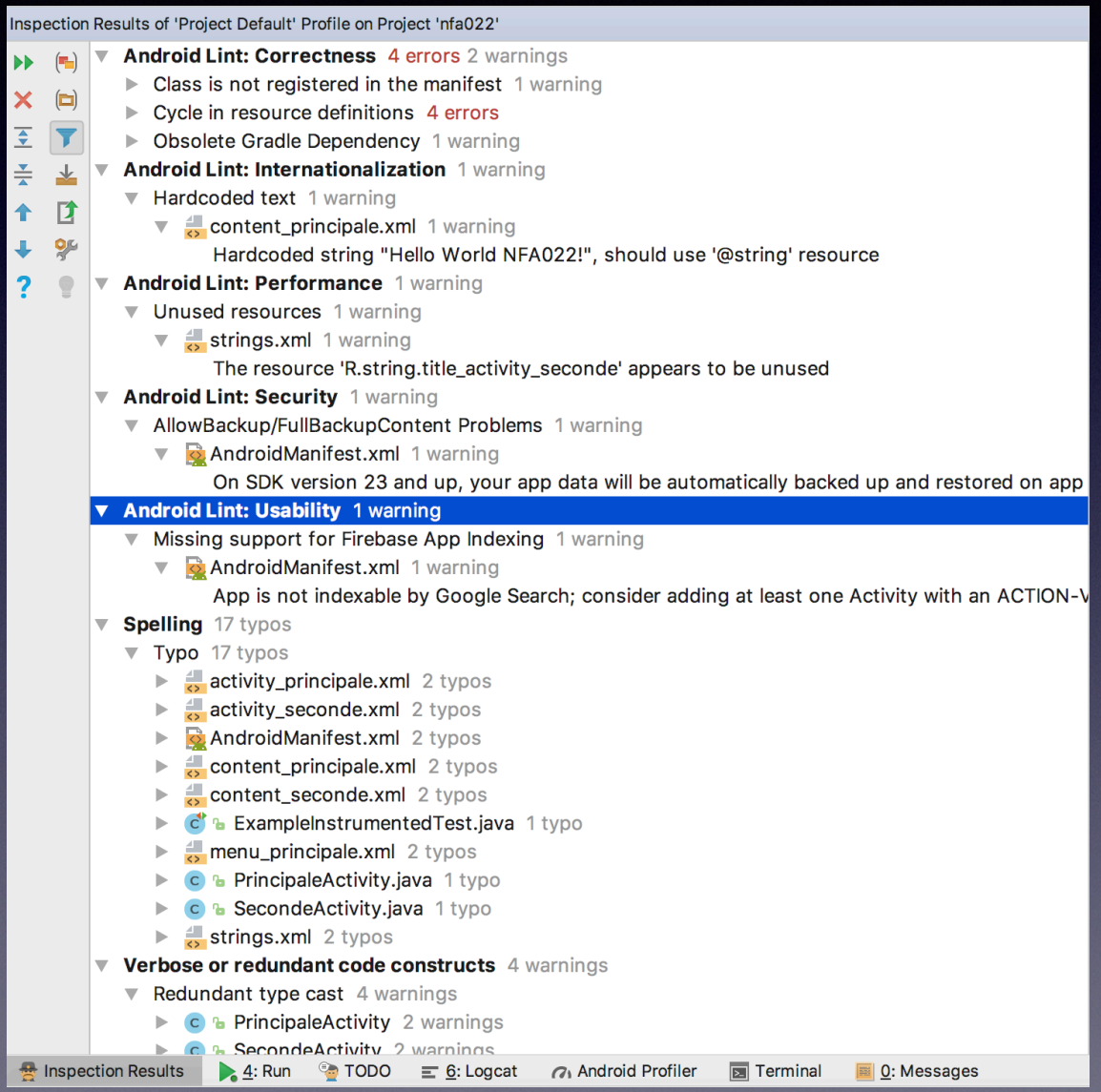
- C'est un outil de ligne de commande, il est disponible dans le ***SDK*** tools à partir de la révision 3.
- Il permet d'optimiser les layouts de l'application.
- Il peut être lancé sur les fichiers de configurations ou les répertoires de ressources pour vérifier les performances.
- Pour exécuter l'outil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

lint

- Successeur de ***layoutopt***, il est disponible dans le ***SDK*** tools à partir de la révision 16.
- L'outil de ***lint*** d'***Android*** est un outil d'analyse statique de code qui permet de
  - vérifier vos fichiers source du projet ***Android***
  - détecter les potentiels bugs
  - proposer des améliorations pour l'optimisation du :
    - la conformité,
    - la sécurité,
    - la performance,
    - la convivialité,
    - l'accessibilité
    - l'interopérabilité
  - Pour exécuter l'outil, lancer : ***lint monProjet***
  - Permet d'analyser les fichiers ***XML*** et ***Java*** dans le répertoire ***monProjet*** et ses sous-répertoires. Le résultat est affiché sur la console.

lint

- Dans **Android Studio** :
  - Analyze > Inspect Code



Compilation

- La compilation sous ***Android*** peut être automatisée en utilisant le plugin ***ADT***.
- ***Android SDK*** utilise aussi ***ANT*** pour automatiser la compilation.
- **ANT** :
  - Ant est écrit en Java
  - Utilisé pour automatiser la construction de projets en langage Java
  - Fonctions implémentées :
    - La compilation (équivalent du make sous Unix),
    - La génération de pages HTML de documentation (Javadoc)
    - La génération de rapports,
    - L'exécution d'outils annexes (checkstyle, FindBugs, etc.),
    - L'archivage sous forme distribuable (JAR, etc.)

Compilation

- La compilation sous ***Android*** peut être automatisée en utilisant **Android Studio**.
- ***Android SDK*** utilise aussi ***Gradle*** pour automatiser la compilation.
- ***Gradle :***
  - Gradle est écrit en Java
  - Utilisé pour automatiser la construction de projets en langage Java
  - Fonctions implémentées :
    - Les mêmes que pour Ant : la compilation, la génération de documentation, la génération de rapports, l'exécution d'outils annexes, l'archivage sous forme distribuable.
    - Gérer et construire les projets Android
  - **Gradle** est intégré à **Android Studio**
    - il est plus souple dans son utilisation
    - il propose une gestion automatique des dépendances
    - il a des fichiers de configuration
    - il s'intègre dans les IDE et notamment dans IntelliJ et AndroidStudio

Développement d'Applications sous Android

Boites à Outils pour Android

Post-it

- Les projet sous Android doivent suivre une structure bien précise.

- Le Plugin ADT d'Eclipse et la SDK incontournable pour le développement sous Android.

- Il existe plusieurs outils de déboggage (LogCat, DDMS, Traceview, etc.) à maitriser si vous développez des applications complexes.