Android

Cours 1 : Introduction à Android / Android Studio

Damien MASSON
damien.masson@esiee.fr

http://www.esiee.fr/~massond

21 février 2017

Références

- `https://developer.android.com` (Incontournable !)
- `https://openclassrooms.com/courses/creez-des-applications-pour-android/` Un tutoriel en français assez complet et plutôt à jour...

Qu'est-ce qu'Android ?

- PME américaine, Android Incorporated, créée en 2003, rachetée par Google en 2005
- OS lancé en 2007
- En 2015, Android est le système d'exploitation mobile le plus utilisé dans le monde ($>$80%)

Qu'est-ce qu'Android ?

Cinq couches distinctes :

1. le noyau Linux avec les pilotes ;
2. des bibliothèques logicielles telles que WebKit/Blink, OpenGL ES, SQLite ou FreeType ;
3. un environnement d'exécution et des bibliothèques permettant d'exécuter des programmes prévus pour la plate-forme Java ;
4. un framework – kit de développement d'applications ;

Android et la plateforme Java

- Jusqu'à sa version 4.4, Android comporte une machine virtuelle nommée Dalvik
- Le bytecode de Dalvik est différent de celui de la machine virtuelle Java de Oracle (JVM)
- le processus de construction d'une application est différent
- Code Java (.java) $\rightarrow$ bytecode Java (.class/.jar) $\rightarrow$ bytecode Dalvik (.dex) $\rightarrow$ interprété
- L'ensemble de la bibliothèque standard d'Android ressemble à J2SE (Java Standard Edition) de la plateforme Java.
- La principale différence est que les bibliothèques d'interface graphique AWT et Swing sont remplacées par des bibliothèques d'Android.

Android Runtime (ART)

- À partir de la version 5.0 (2014), l'environnement d'exécution ART (Android RunTime) remplace la machine virtuelle Dalvik.
- Avec ART, contrairement à la machine virtuelle java, les fichiers .apk ne sont plus lancés directement, mais décompressés, compilés et lancés
- les applications prennent ainsi plus de place (+20 %),
- les gains en performance et en autonomie des batteries sont conséquents (+20 à 30 %).

Pour le développeur... ça ne change rien !

Processus de développement

Le développement pour Android s'effectue avec un ordinateur personnel sous Mac OS, Windows ou Linux en utilisant

- Le JDK de la plate-forme Java
- Des outils qui permettent
  - de manipuler le téléphone ou la tablette,
  - de la simuler par une machine virtuelle,
  - de créer des fichiers .apk,
  - de déboguer les applications et d'y ajouter une signature numérique.
- Ces outils sont mis à disposition via un plugin Eclipse (ADT) jusqu'en 2014,
- Puis via un IDE maison appelé AndroidStudio basé sur IntelliJ IDEA

Déploiement

Version	Nom de code	Date de sortie	API	%
2.3.3 - 2.3.7	Gingerbread	12/2010	10	1,0 %
4.0.3 - 4.0.4	Ice Cream Sandwich	10/2011	15	1,0 %
4.1.x	Jelly Bean	07/2012	16	4,0 %
4.2.x	Jelly Bean	09/2012	17	5,7 %
4.3.x	Jelly Bean	07/2013	18	1,6 %
4.4-4.4.4	KitKat	10/2013	19	21,9 %
5.0-5.0.2	Lollipop	11/2014	21	9,8 %
5.1.x	Lollipop	03/2015	22	23,1 %
6.0	Marshmallow	10/2015	23	30,7 %
7.0	Nougat	08/2016	24	0,9 %
7.1	Nougat	10/2016	25	0,3 %

Qu'est-ce que Java ?

C'est un ensemble de spécifications :

- une pour le langage de programmation
  `http://java.sun.com/docs/books/jls/`
- une pour le bytecode et la machine virtuelle
  `http://java.sun.com/docs/books/jvms`
- d'autres pour plein de choses (ex. RTSJ pour le temps réel, voir le site des JSR)

plus des API.

Ce n'est pas :

- l'implantation de Sun/Oracle cad Hotspot (mais qui respecte la spécification)

Dalvik n'est donc pas une JVM Java, puisque ne respecte pas la spécification pour le bytecode et la JVM. Par contre elle interprète du bytecode (pas Java) qui peut être obtenu par compilation à partir d'un programme Java.

Qu'est-ce qu'une Machine Virtuelle ?

- "programmer", c'est résoudre un problème par des calculs
- il existe de nombreux *modèles de calculs* :
  - Digitals/Analogiques,
  - Séquentiels/Parallèles,
  - Automatiques/Supervisés,
  - Fonctionnels/Impératifs/Logiques/Concurrent...
- pour chaque modèle de calcul, il existe une *machine abstraite* qui décrit son environnement d'évaluation
- la plupart des machine abstraites n'ont pas de réalisation physique (exception : la machine de Turing)
- mais un programme peut émuler une machine abstraite : ce programme est une machine virtuelle

Digressions sur les machines de Turing

Une machine de Turing est un modèle abstrait du fonctionnement des appareils mécaniques de calcul (1936).

La mise en œuvre concrète d'une machine de Turing est réalisée avec les éléments suivants :

- Une bande (infinie) de cases contenant des symboles.
- Une tête de lecture en face d'une case
- Un état courant (p.ex. un numéro d'état)
- Un programme (fini), qui à tout état et symbole lu, associe un nouvel état,
- un symbole à écrire, et un mouvement (G/D) de la tête de lecture

Digressions sur les machines de Turing

- Les ordinateurs modernes (ou pas) sont des machines de Turing :
  - bande = mémoire
  - état = pointeur de code
  - programme = câblage CPU

- **Thèse** de Church-Turing : tout ce qui se calcule peut se calculer via une machine de Turing
- Une seule machine suffit
- car une MdT peut calculer le comportement d'une MdT
- autrement dit une Mdt peut émuler une MdT
- donc un ordinateur peut calculer tout se qui se calcule.

- voir cours de calculabité...

Architecture des ordinateurs *pour les nuls*

Machines pas virtuelles

- Composants d'un ordinateur :
  - Processeur : exécute les instructions
  - Mémoire principale : stocke les programmes et les données
  - Mémoire secondaire : idem de façon persistante
  - Périphérique d'entrée/sortie : communication avec l'utilisateur

- Ce qui suit décrit une simplification très approximative du fonctionnement d'une machine moderne.
- Plus : vos cours d'archi, ou (par exemple) `http://www-gtr.iutv.univ-paris13.fr/Cours/Mat/Architecture/Cours/`

Processeur

Trois composantes à l'intérieur du processeur :

- L'unité de contrôle coordonne l'avancée des calculs.
- L'unité arithmétique effectue les calculs.
- Les registres servent de mémoire à accès très rapide.

Différents types de registres en fonction de leur utilisation :

- les registres d'adressage ;
- les registres de données d'entrées ;
- les registres de résultats.

Il existe souvent des registres spéciaux :

- IR : un registre qui stocke l'instruction courante.
- PC : un registre qui stocke l'adressage de l'instruction courante dans le code.

La mémoire

- La mémoire est exclusivement formée de 0 et de 1, les bits.
- Nous choisissons une interprétation de cette mémoire.
- Pour faciliter la définition de leur interprétation et leur manipulation, on a choisi de grouper ces bits en paquets de 8 bits, appelés octets (bytes).
- Pour traiter plus rapidement les informations, un processeur groupe ces octets en mots. La taille des mots varie d'un processeur à l'autre : 4 octets (32 bits) ou 8 (64 bits) ou ...

Mémoire et données

- Une information est localisée dans la mémoire par une adresse.
- La plupart du temps, chaque cellule mémoire contient un octet.
- Un mot doit donc être réparti sur plusieurs octets consécutifs.
- Il existe plusieurs façons d'agencer ces octets : little endian (i386, ...) ou big endian (SPARC, ...) ou ...
- Le processeur lit le contenu d'une adresse en le stockant dans un registre.
- Le processeur écrit dans une adresse en y stockant le contenu d'un registre.
- La mémoire primaire est rapide mais s'efface lorsque la machine est éteinte.
- La mémoire secondaire est lente mais persiste même si la machine est éteinte.

Mémoire et programmes

- Les programmes sont stockés dans la mémoire secondaire.
- Pour les exécuter (rapidement), ils sont chargés en mémoire primaire.
- Une instruction est représentée par une séquence de bits qui suit généralement l'organisation : `opcode - data`
- La taille d'une instruction est généralement proportionnelle à la taille des mots du processeur.
- Le code de l'opération (`opcode`) est un indice faisant référence à la table d'instructions du processeur (`add`, `mul`, `push`, ...).
- En fonction de ce code, la suite de l'instruction est décodée : ce sont des données réparties en deux catégories, les opérandes et les adresses.

Le cycle Récupère-Décode-Exécute

Le cycle *Récupère-Décode-Exécute* (Fetch-Decode-Execute) est l'activité principale d'un processeur. En gros :

1. Phase *Récupère* : Le processeur détermine la prochaine instruction à partir de l'adresse contenue dans le registre PC et la stocke dans le registre IR. Dès que IR est mis-à-jour, PC est incrémenté pour la prochaine itération.
2. Phase *Décode* : Le processeur interprète la partie data de l'instruction en fonction de la partir opcode
3. Phase *Exécute* : Le processeur effectue l'opération

La vitesse de l'horloge (la fréquence du processeur) correspond au nombre de cycles effectués par unité de temps.

Avantages des Machines Virtuelles

Une machine virtuelle est donc un programme qui est exécuté par une machine (virtuelle ou pas). Elle n'a pas d'existence physique, cela permet :

- de s'abstraire totalement du matériel (de l'électronique notamment), et de ne raisonner que sur les programmes.
- de contrôler l'exécution d'un programme : la machine virtuelle peut observer le programme, vérifier son comportement et même intervenir en cas d'erreur du programme.
- la machine virtuelle étant un programme, elle peut être manipulée comme tel (être mise à jour, échangée entre des terminaux etc.)

Inconvénients des Machines Virtuelles

- la couche supplémentaire d'interprétation du code (puisqu'on émule une machine abstraite, il y a nécessairement transcription du code) rend difficile l'écriture d'une machine virtuelle efficace (mais c'est possible).
- la difficulté de l'analyse d'un programme avant son évaluation rend difficile l'écriture d'une machine virtuelle sûre (mais c'est possible).

Inconvénients des Machines Virtuelles

- la couche supplémentaire d'interprétation du code (puisqu'on émule une machine abstraite, il y a nécessairement transcription du code) rend difficile l'écriture d'une machine virtuelle efficace (mais c'est possible).

Pour améliorer l'efficacité, on utilise un bytecode proche du code de l'architecture cible, pour réduire la tâche de l'interpréteur. On peut aussi compiler le code juste avant l'évaluation (JIT).

- la difficulté de l'analyse d'un programme avant son évaluation rend difficile l'écriture d'une machine virtuelle sûre (mais c'est possible).

Inconvénients des Machines Virtuelles

- la couche supplémentaire d'interprétation du code (puisqu'on émule une machine abstraite, il y a nécessairement transcription du code) rend difficile l'écriture d'une machine virtuelle efficace (mais c'est possible).
- la difficulté de l'analyse d'un programme avant son évaluation rend difficile l'écriture d'une machine virtuelle sûre (mais c'est possible).

le bytecode est une donnée exécutable, il doit donc être manipulé avec précaution : soit authentifié, soit analysé statiquement, soit exécuter d'une une sandbox (politique de sécurité pour l'accès aux périphériques sensibles)

La Machine Virtuelle Java (JVM)

- La JVM a été initialement spécifiée par Sun MicroSystem (racheté par Oracle) pour exécuter le bytecode produit par les compilateurs Java (qui respecte la spécification du langage).
- Sa spécification est publique : `http://java.sun.com/docs/books/vmspec/` et il en existe de (très) nombreuses implémentations :

Azul VM - CEE-J - Excelsior JET - J9 (IBM) - JBed - JamaicaVM - JBlend - JRockit - Mac OS Runtime for Java (MRJ) - MicroJvm - Microsoft Java Virtual Machine - OJVM - PERC - Blackdown Java - C virtual machine - Gemstone - Golden Code Development - Intent - Novell - NSIcom CrE-ME - HP ChaiVM MicrochaiVM - HotSpot - AegisVM - Apache Harmony - CACAO - Dalvik - IcedTea - IKVM.NET - Jamiga - JamVM - Jaos - JC - Jelatine JVM - JESSICA - Jikes RVM - JNode - JOP - Juice - Jupiter - JX - Kaffe - leJOS - Maxine - Mika VM - Mysaifu - NanoVM - SableVM - Squawk virtual machine - SuperWaba - TinyVM - VMkit - Wonka VM - Xam

- La spécification laisse une importante liberté d'implémentation.

Modèle de calcul

- C'est une machine à pile
- pensée pour la programmation objet, concurrente et mobile
  - instructions pour appels de méthodes, threads et chargement dynamique de code avec vérification
- adapté pour la compilation d'autres langages que Java (compilateurs vers bytecode pour ADA, AWK, C, COMMON LISP, FORTH, RUBY, LUA, OCAML...)

Modèle de calcul

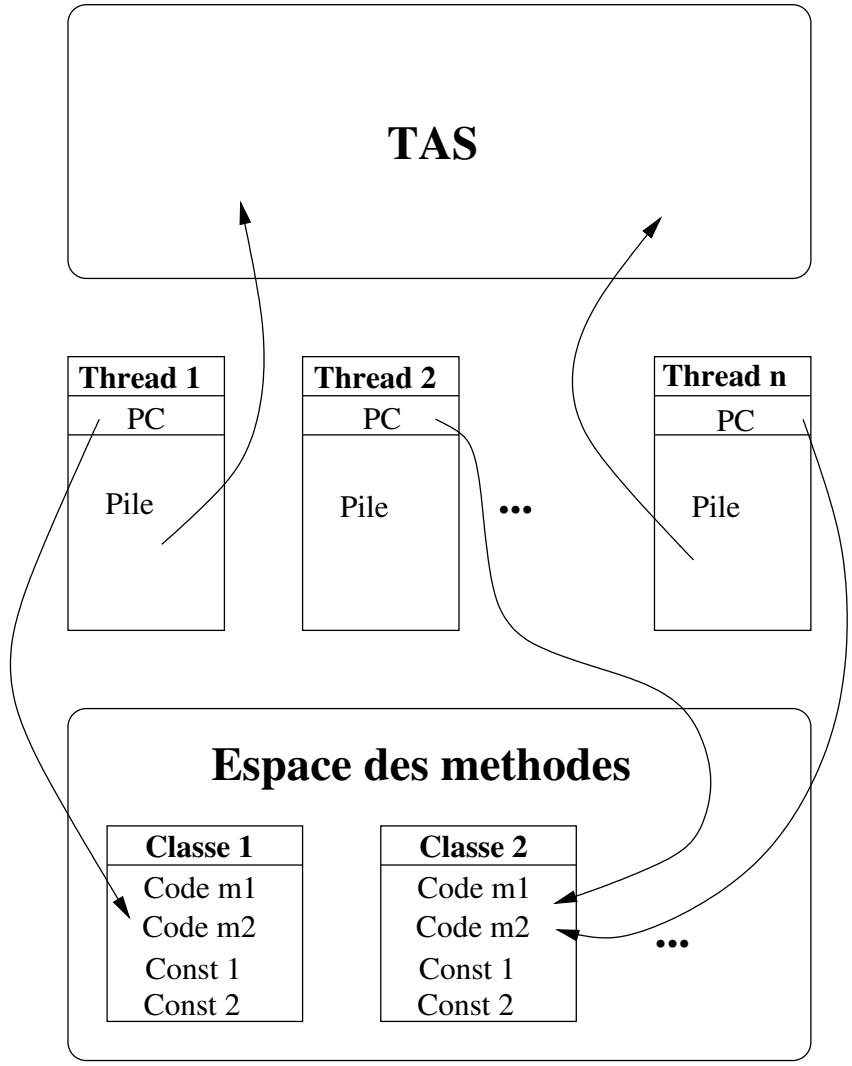
- C'est une machine à pile

- Quand on lit une opérande, elle est empilée ;
- Quand on lit un opérateur, les opérandes nécessaires sont dépilées, l'opérateur appliqué et le résultat empilé ;
- le résultat du calcul est ce qui reste dans la pile à la fin.

- pensée pour la programmation objet, concurrente et mobile
  - instructions pour appels de méthodes, threads et chargement dynamique de code avec vérification
- adapté pour la compilation d'autres langages que Java (compilateurs vers bytecode pour ADA, AWK, C, COMMON LISP, FORTH, RUBY, LUA, OCAML...)

Composants de la JVM

- Le tas et l'espace des méthodes sont partagés entre les threads mais les piles et les PC sont privés
- Toutes ses composantes évoluent durant l'exécution
- Les données au format `.class` permettent de peupler (dynamiquement) l'espace des méthodes
- dans HotSpot, l'espace des méthodes fait partie du tas, mais ce n'est pas imposé par la spécification



En ajoutant un chargeur de classes, un GC (même pas obligatoire) et l'interpréteur vous avez une JVM.

Les types manipulés

La JVM manipule deux grandes familles de données :

- les données de types primitifs :
  - les valeurs numériques : entières ou à virgule flottante ;
  - les booléens ;
  - les adresses de code (non modifiables par le programme).
- Les références qui sont des pointeurs vers des données allouées dans le tas (des instances de classe ou des tableaux).

La façon dont ces données sont représentées en mémoire est spécifiée ! (nombre de bits, ordre etc.)

L'espace des méthodes

Pour chaque classe, l'espace des méthodes contient :

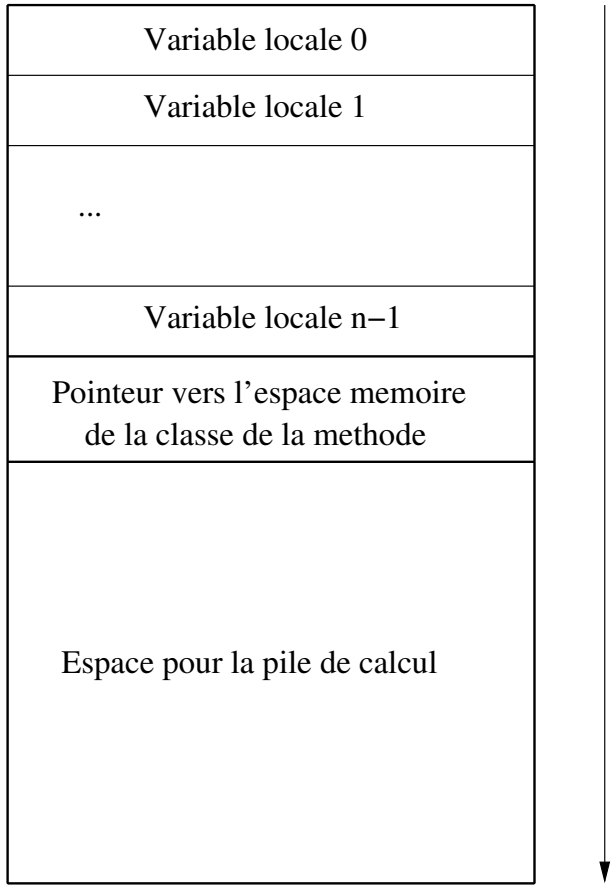
- un ensemble de constantes ;
- des champs de classes partagés (les champs static) ;
- des données liées aux méthodes ;
- le code des méthodes et des constructeurs ;
- le code de méthodes spéciales pour l'initialisation des instances de classes et de type d'interface (blocks static et constructeurs).

La pile

- La pile privée sert à stocker des blocs d'activation.
- Ce sont des espaces mémoires temporaires pour les variables locales et les résultats temporaires.
- La pile privée est aussi utilisée pour passer l'adresse de retour d'une méthode ainsi que ses arguments effectifs.
- On accède à un seul bloc d'activation à la fois.

L'exécution du code

- À chaque fois qu'une méthode est invoquée, un bloc d'activation est empilé.
- Quand l'exécution de la méthode est terminée, ce bloc est dépilé.



Taille du bloc d'activation

- Le calcul des tailles des différentes composantes du bloc d'activation est effectué par le compilateur.
- Une variable locale a une taille de 32 bits.
- Les données de type long ou double utilisent deux variables locales.
- On accède aux variables locales via leur position dans le bloc d'activation.
- Ces indices débutent à 0.
- Les variables locales sont utilisées pour stocker les arguments effectifs des appels de méthode. Par convention, le premier argument (d'indice 0) contient toujours la référence vers `this`, c'est-à-dire l'instance de l'objet dont on exécute la méthode.
- La taille maximale de pile nécessaire à l'exécution d'un code donné est calculable.

Le jeu d'instuction

- Les instructions de la JVM sont typées.
- Le nom de chaque opération est préfixée par une lettre indiquant le type des données qu'elle manipule :
  - i : int
  - l : long
  - s : short
  - b : byte
  - c : char
  - f : float
  - d : double
  - a : reference
- Par ailleurs, les opcodes sont stockés sur un octet : il y en a donc au plus 256 !
- Référence : `http://java.sun.com/docs/books/jvms/second_edition/html/Instructions.doc.html`

Le format .class

voir la spécification... ou la page wikipedia en francais

Les outils

- javap -c -verbose
- jasmin
- note : il est possible d'écrire du bytecode valide qui ne peut pas être obtenu par compilation d'un programme Java valide !

Pourquoi pas une JVM dans Android ?

Google a privilégié un autre format de bytecode, packagé dans des fichiers .dex, et un autre type de machine virtuelle : une machine à registres.
Pourquoi ?

Java Micro Edition (JME) ?

J2ME :

- JSR 68 puis...
- framework Java spécialisé dans les applications mobiles (en gros : téléphones/PDA et décodeurs multimédia)
- KVM + configuration + profil
- Java 1.3
- Contrairement à J2SE et J2EE : pas de "classpath exception" sur la licence GPLv2 !
- environnement un peu trop hétérogène
- lenteur d'évolution par le JSR, plus veto possible de Sun

Java Community Process (JCP) et Java Specification Request (JSR) ?

JCP :

- coordonner l'évolution du langage Java et des technologies qui lui sont associées
- Le JCP émet des Java Specification Requests (JSR)

JSR :

- Un early draft, puis un draft puis une release,
- avec une implantation et un kit de test
- Processus normalisé (le JCP fait l'objet d'une JSR - la 215)

Pourquoi pas une nouvelle JSR pour les smartphone ?

Finalement, licence d'Android

Android (et donc Dalvik) est distribué sous *Apache licence*

- libre et opensource, ce qui permet de
  - exécuter le programme pour tout les usages
  - étudier le fonctionnement du programme (code source livré)
  - modifier le programme
  - redistribuer des copies du programme ou de ses modifications (y compris à la vente)
- non copyleft : on peut distribuer un système Android modifié sous une autre licence (y compris propriétaire)

Android : Rappels

Dalvik vs Java VM

- Dalvik est une machine virtuelle à registres
- Dalvik est conçue pour faciliter l'exécution concurrente de plusieurs instances de la VM
- Car avec Dalvik, 1 processus = 1 VM
- Dalvik interprète du *bytecode* au format .dex

1 Processus = 1 VM

Pourquoi ?

- méthode classique *sandboxing*
- garanti l'isolation des processus entre eux et avec le système
- une activité ou un service plante ? un seul processus plante
- la VM plante ? un seul processus disparaît

Cela impose :

- l'exécution concurrente de plusieurs VM doit être efficace
- la création d'une nouvelle instance de la VM doit être rapide

Zygote et mémoire en lecture seule

- Tant que du code est accédé en lecture seule, il reste partagé entre les instances de la VM
- La duplication se fait au dernier moment en cas d'écriture
- Permet d'accélérer la création d'une nouvelle VM (moins de copie)

Zygote et mémoire en lecture seule

- une VM/processus spéciale, appelée Zygote, est démarrée en même temps que le système
- les classes de base d'Android sont pré-chargées et pré-initialisées dans cette VM
- c'est cette VM qui charge les nouvelles classes
- si une application tente de modifier des données partagées, il y a copie de ces données du Zygote vers la VM fille qui abrite cette application
- c'est ce processus qui se fork pour créer une nouvelle VM "prête à l'emploi"
- attention GC

Machine à registres

Machine abstraite équivalente à une machine de Turing

- possède une infinité de *registres*
- chaque registre peut stocker un entier arbitrairement long
- chaque registre dispose d'une adresse unique
- l'ensemble des instructions agissent sur les registres

Exemple : un processeur !

Machines à registres vs machine à piles

- dispatch (chargement)
- fetch (décodage)
- execute (exécution)

Dispatch

Pour charger une instruction, c'est pareil, mais ...

Exemple : $a = b + c$

stack machine

```
ILOAD c ; ILOAD b ; IADD ;
ISTORE a ;
```

register machine

```
IADD a,b,c ;
```

- moins d'instructions pour faire la même chose ? pas forcément, si on peut optimiser le code pour que les opérandes soient systématiquement au sommet de la pile, ça revient au même
- en pratique, il n'y a pas une infinité de registre, donc il y aurait des instructions de chargement/déchargement en mémoire qui seraient cachées ?
- pas vrai car en codant les adresses des registres sur 1 octet on en a 256. En pratique il n'y a pas de méthode Java qui utilisent 256 variables locales !
- mais instructions plus longues à décoder ?

Fetch

- avec une machine a pile, la plupart des instructions agissent sur la pile : les opérandes sont donc implicites
- donc le décodage d'une instruction de machine à registre serait plus couteux ? deux choses : trouver l'adresse des opérandes, et récupérer leurs valeurs
- pour trouver les opérandes, avantages aux machines à pile
- en revanche, les registres comme la pile sont codés avec un tableau, donc égalité pour aller chercher les opérandes ?
- sauf qu'avec une machine à pile, on peut mettre en cache (dans un registre) le haut de la pile...
- mais 1 seule instruction permet de récupérer et d'opérer sur une valeur avec les registres
- alors que pour les machine à piles, les opérations sur la pile sont des instructions séparées, qui peuvent donner lieu a de vrais load/store...

Exécute

- exécuter l'instruction à le même cout, quelque soit la représentation intermédiaire
- mais dans le modèle à registres, on ne détruit pas les résultats intermédiaires
- qui peuvent plus facilement être ré-exploité en cas de code redondant ?
- en C, éliminer toutes les redondance dans standard SPECint 95 C benchmarks réduit les instructions uniquement de 3%

Machines à registres vs machine à piles

- dispatch (chargement) : Registres 1 / Piles 0
- fetch (décodage) : Registres 0 / Piles 1
- execute (exécution) : match nul

$$T_{VRM} = T_{VSM} - \#dispatches \times T_{dispatch} + \#fetches \times T_{fetch}$$

Machines à registres vs machine à piles

D'après Gregg et al. (2005) traduire du bytecode Java en bytecode pour machine à registres :

- réduit le nombre d'instructions de 35%
- augmente la taille du bytecode de 44%
- augmentant le nombre moyen de chargements de 45%

Comme le coût du chargement est faible par rapport à celui du dispatch (qui est fait pour chaque instruction), le bilan est positif

D'après D. Bornstein (Google) [1] :

- .dex réduirait de 47% le nombre d'instructions
- augmenterait la taille du bytecode de 25%
- ce qui augmenterait le nombre de chargements de 1,07%
- et diminuerait le temps d'exécution de 32%

1. mais je n'ai pas trouvé de publication autre que la vidéo de présentation de Dalvik...

Le format Dex

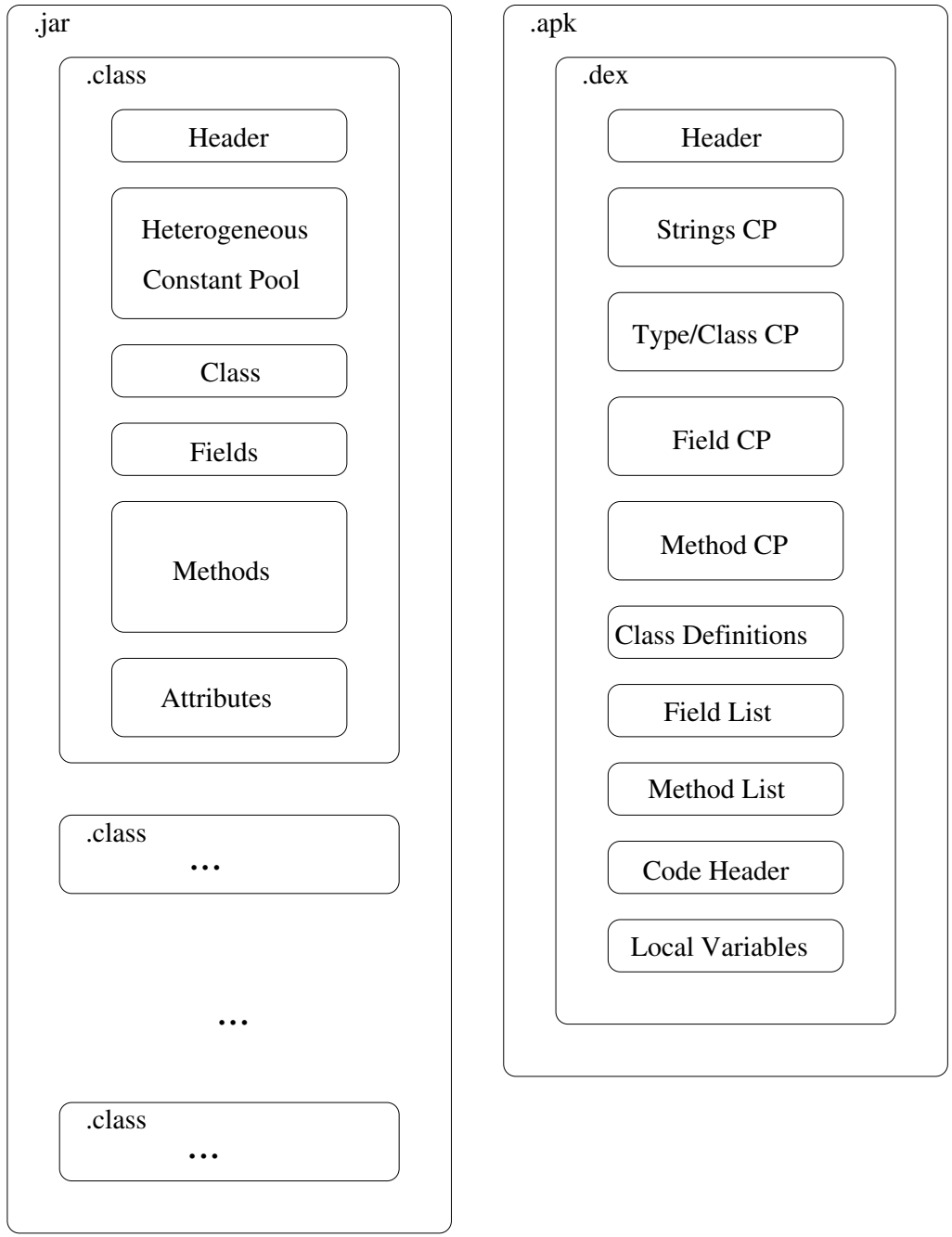
Chaine de compilation pour Android :

$$.java \xrightarrow{javac} .class \xrightarrow{dx} .dex$$

- 1 .java peut donner plusieurs .class car 1 classe = 1 .class
- une application = un .dex
- + manifest etc. = un .apk

- But : partager ce qui est partageable entre les classes

.class/.jar VS .dex/.apk



.class/.jar VS .dex/.apk

- Partage des constantes du CP entre toutes les classes
- Constant Pool typés, permet une plus grande granuarité

```
(Ljava/lang/String;Ljava/lang/Object;)Ljava/lang/String;
(Ljava/lang/String;Lcom/example/Foo;)Ljava/lang/String;
```

- Mais cela multiplie les niveaux d'indirection...

Sécurité

- La plateforme Android gère déjà les aspects de sécurité, la machine virtuelle n'a pas à s'en occuper
- Seules les applications installées peuvent s'exécuter
- l'isolation des processus dans leurs propres VM les protège des autres
- et protège le système
- les permissions spéciales sont demandées dans le manifest

Conclusion

- Ceci est la fin des cours sur l'intérieur d'android
- ayez la curiosité de récupérer les sources et d'y jeter un oeil