Dalvik Dex Format

VS

Java Bytecode

Xposé 2010 - 2011

Plan

- **Introduction :**
  - Le format Dex
  - La machine virtuelle Dalvik
- **Comparatif :**
  - La différence en Dalvik et les autres JVMs
  - Les spécificités du format Dex

Mise en situation

- **Pour faciliter la compréhension du sujet :**
  - **Auditoire => Etudiant IG2K**
  - **Orateur => Génie de l'idée**

Introduction

Le ByteCode Java

- Code intermédiaire exécutable par une JVM
- Résulte du la compilation d'un programme Java
- Permet l'exportation d'un binaire Java sur différentes architectures systèmes

Introduction au format Dex

- Dex (Dalvik EXecutable)
- Format spécifique de fichier d'exécution Java :
  - Contient du ByteCode spécifique
- Est une « transformation » du format d'exécution normal Java
- Spécifique à la plateforme Android

L'efficacité du format Dex

- Efficacité sur appareils mobiles ≠ Efficacité sur PC
- Design du format Dex fait pour amélioré :
  - Gestion de la mémoire
  - Rapidité processeur

La machine Virtuelle Dalvik

- LA machine virtuelle d'Android
- N'est pas une machine virtuelle Java

- Est au cœur du système d'Android

La machine Virtuelle Dalvik

- Convient à une architecture :
  - Possédant peu de RAM
  - Alimentée par batterie
  - Avec un microprocesseur lent
  - Ne possédant pas de mémoire swap

Récapitulatif de l'introduction

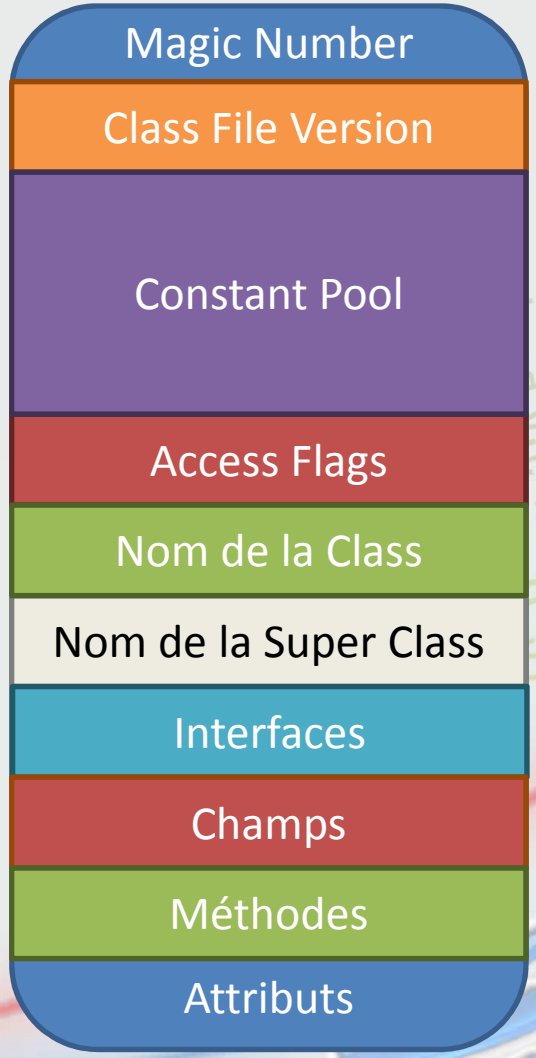
- Le bytecode => Code intermédiaire permettant l'interprétation de programme sur différente plateforme
- Fomat Dex => Transformation du format d'exécution Java pour Android
- Dalvik => Machine virtuelle permettant d'interpréter le format Dex

Comparaison

Fichier .class VS Fichier .dex

Les entrailles des fichiers .class

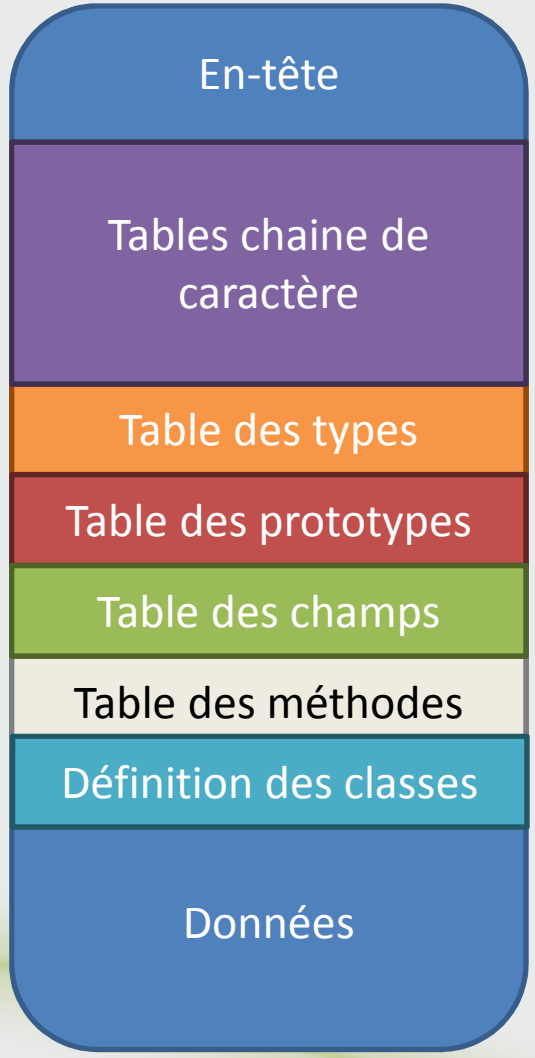
- Un fichier .class par Class déclarée dans le programme
- Contient 10 sections :



Fichier .class VS Fichier .dex

Les entrailles des fichiers .dex

- Un fichier .dex pour TOUT le programme
- Contient 8 sections :



Fichier .class VS Fichier .dex

Les types de données fondamentaux

- **Fichier .dex** (little endian)
  - Signé ou non :
  - Pour les valeurs 32 bits
    => codage LEB 128
- **Fichier .class** (big endian)
  - Non signé :

Fichier .class VS Fichier .dex

Obtenir un fichier .class

- Pour une compilation simple :
  - En ligne de commande => javac
  - Graphiquement => via un IDE

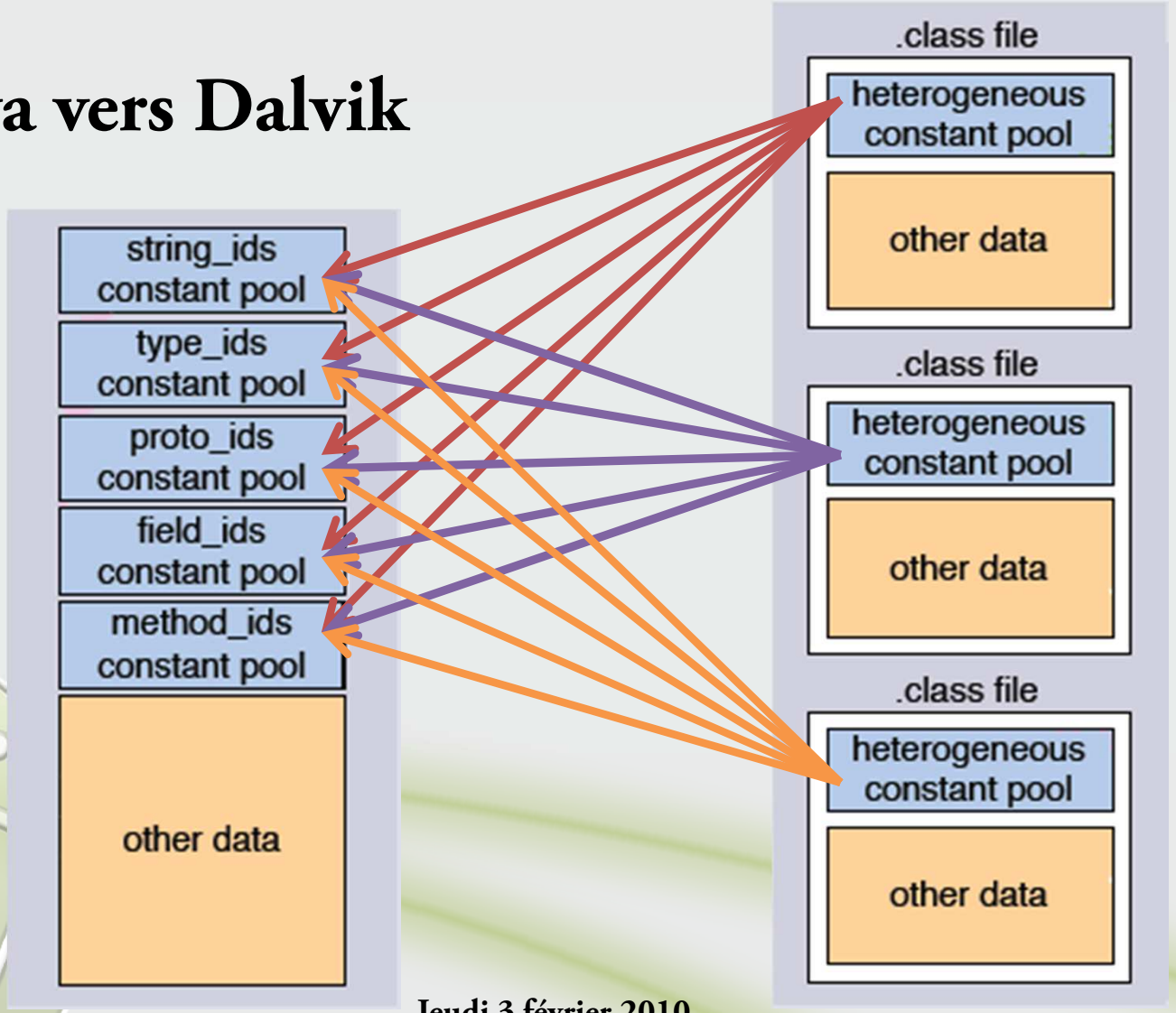
Fichier .class VS Fichier .dex

Obtenir un fichier .dex

- **Pour une compilation simple :**
  - **Compiler le programme en java classiquement**
  - **Utiliser l'utilitaire nommé 'dx'**

Fichier .class VS Fichier .dex

De Java vers Dalvik



Fichier .class VS Fichier .dex

Les avantages recherchés par le format Dex

- Minimiser les besoins en mémoire
- Rappel Android :
  - RAM disponible après démarrage des services : 20 Mo (à partager avec les autres processus)
- Atteint par :
  - Minimisation des répétions
  - Pool par type

Fichier .class VS Fichier .dex

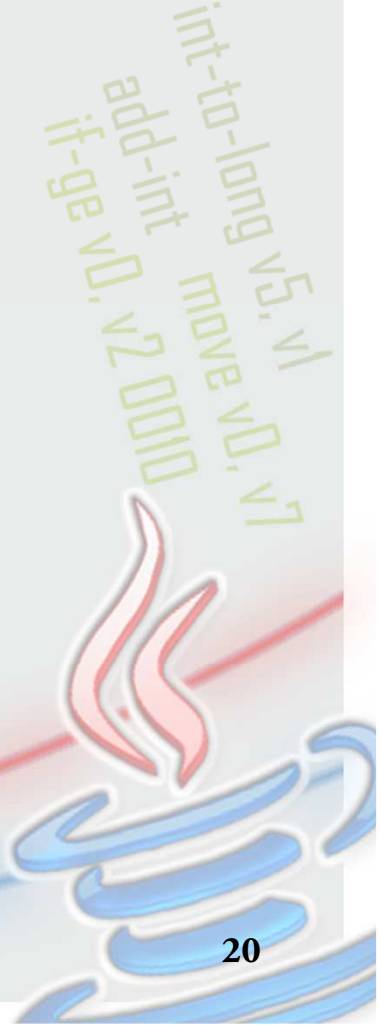
La preuve par les chiffres

- **Les librairies système :**
  - Fichier jar compressé => 50%
  - Fichier dex => 48%
- **Application Réveil :**
  - Fichier jar compressé => 52 %
  - Fichier dex => 44%

Java VM VS Dalvik VM

Les autres possibilités pour Java sur Android

- Symbian (Nokia) :
  - CLDC Hotspot
- IPhone : Impossible
- IPhone avec Jailbreak :
  - JamVM
- Autre :
  - KVM
  - MicroJVM
  - Jazelle & ThumbEE



Java VM VS Dalvik VM

La différence de Dalvik

- Register Machine (Alors que les JVM sont des Stack Machine)

- Ce qui est recherché :
  - Éviter les accès mémoire inutiles

Java bytecode VS Dex bytecode

Le code source d'une méthode

```
public static long sumArray(int[] arr) {
    long sum = 0;
    for (int i : arr) {
        sum += i;
    }
    return sum;
}
```

Java bytecode VS Dalvik bytecode

Du côté Java

- Les stats :
  - 25 octets
  - 45 lectures
  - 16 écritures

```
0000: lconst_0
0001: lstore_1
0002: aload_0
0003: astore_3
0004: aload_3
0005: arraylength
0006: istore 04
0008: iconst_0
0009: istore 05
000b: iload 05
000d: iload 04
000f: if_icmpge 0024
0012: aload_3
0013: iload 05
0015: iaload
0016: istore 06
0018: lload_1
0019: iload 06
001b: i21
001c: ladd
001d: lstore_1
001e: iinc 05, #+01
0021: goto 000b
0024: lload_1
0025: lreturn
```

Java bytecode VS Dalvik bytecode

Du côté Dalvik

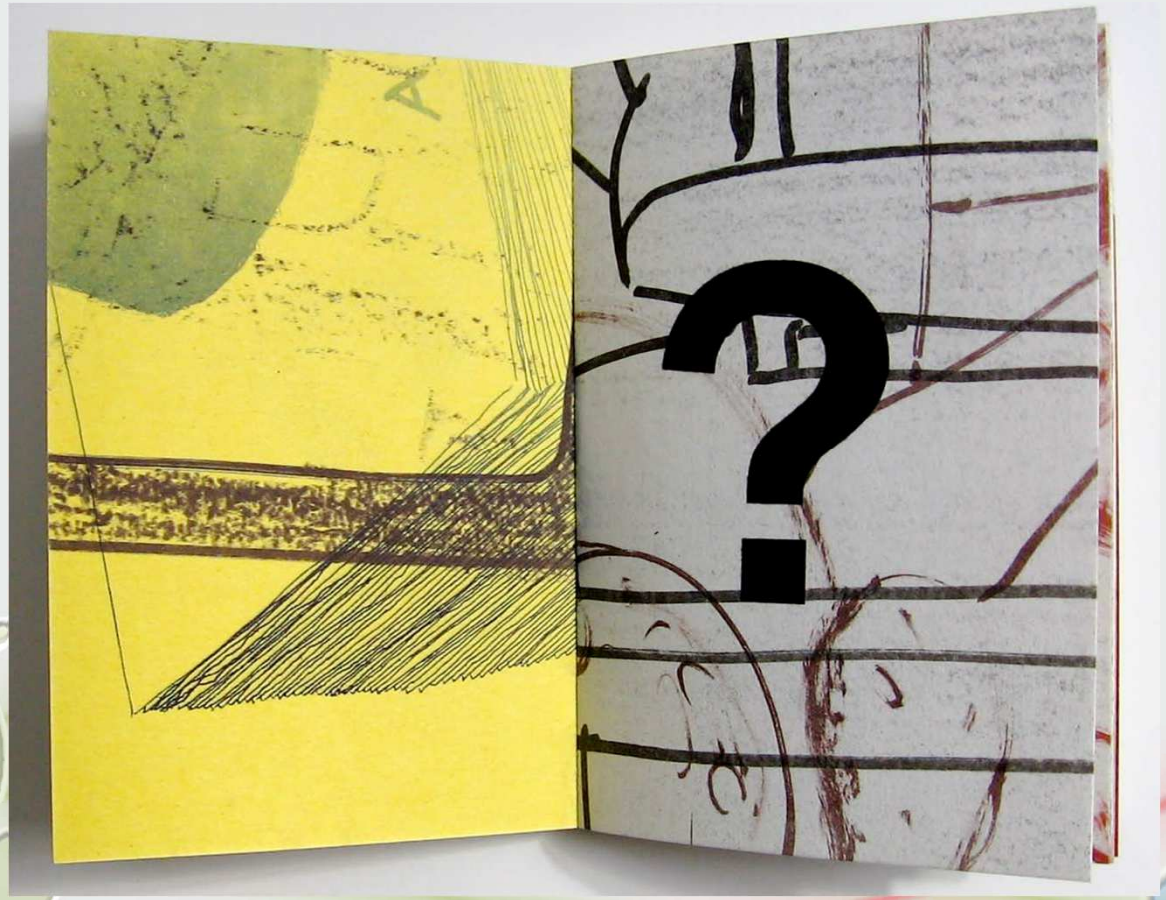
- Les stats :
  - 18 octets
  - 19 lectures
  - 6 écritures

```
0000: const-wide/16 v0, #long 0
0002: array-length v2, v8
0003: const/4 v3, #int 0
0004: move v7, v3
0005: move-wide v3, v0
0006: move v0, v7
0007: if-ge v0, v2, 0010
0009: aget v1, v8, v0
000b: int-to-long v5, v1
000c: add-long/2addr v3, v5
000d: add-int/lit8 v0, v0, #int 1
000f: goto 0007
0010: return-wide v3
```

Conclusion

- Format Dex correspond à ce pourquoi il est conçu
- Ajoute une certaine complexité pour les développeurs
- Ne supporte pas toutes les API Java

Questions



Nimp

- **Pour mobile -> J2ME**
- CLDC -> Connected Limited Device Configuration The Connected Limited Device Configuration (CLDC) is a specification of a framework for Java ME applications describing the basic set of libraries and virtual-machine features that must be present in an implementation. The CLDC is combined with one or more profiles to give developers a platform for building applications on embedded devices with very limited resources such as pagers and mobile phones.
- MIDP -> Mobile Information Device Profile (MIDP) is a specification published for the use of Java on embedded devices such as mobile phones and PDAs. MIDP is part of the Java Platform, Micro Edition (Java ME) framework and sits on top of Connected Limited Device Configuration (CLDC), a set of lower level programming interfaces. MIDP was developed under the Java Community Process. The first MIDP devices were launched in April 2001.