NFA022 : Programmation des smart-phones et tablettes tactiles

Jean-Ferdy Susini
Maître de Conférences - CNAM
Département Informatique

le cnam

Paris, 12 févr. 2013

Évolution des “smart-devices”

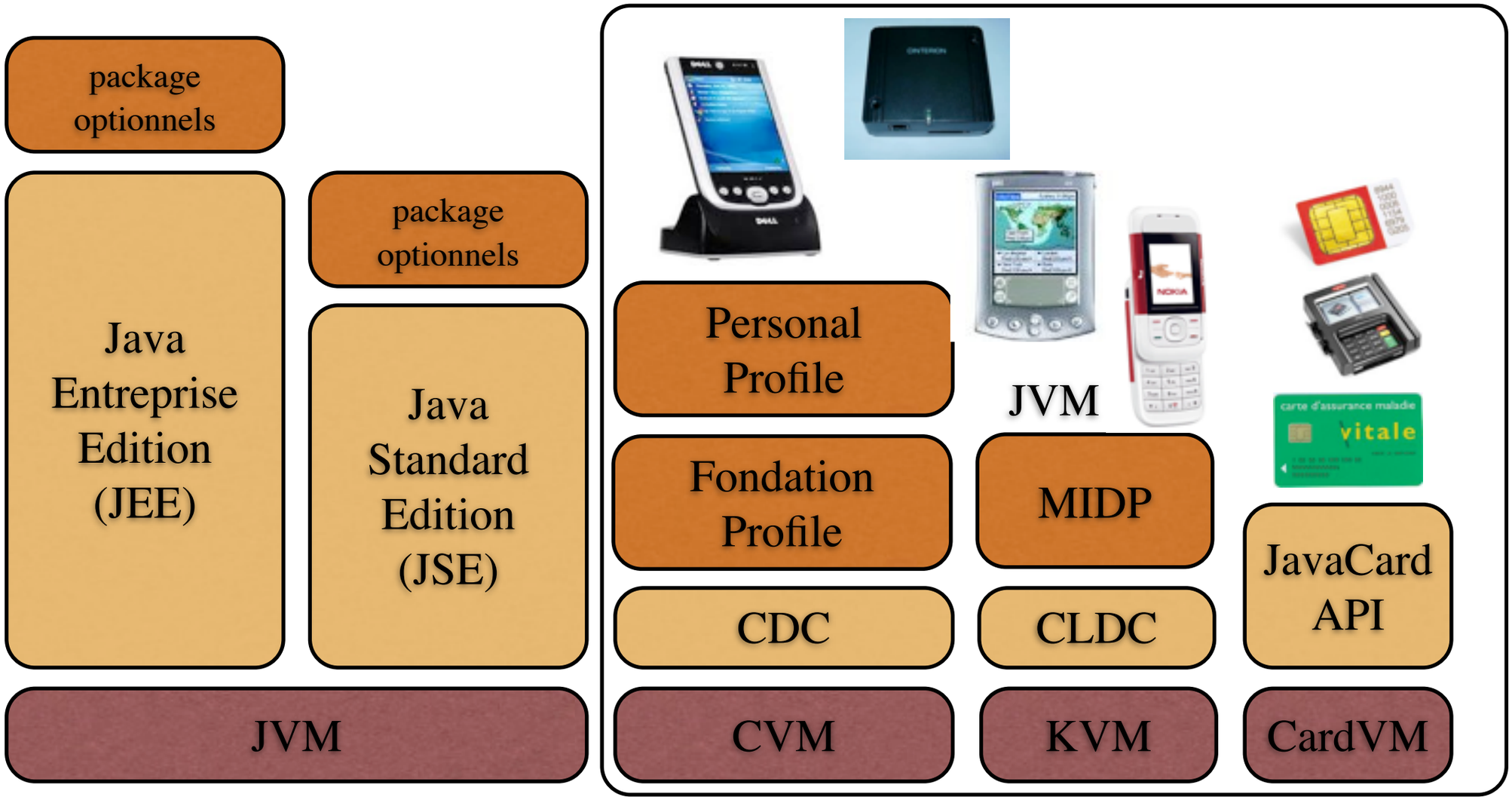
Sources : Android.com, Wikipedia, et différents autres sites…

le cnam

Un peu d'histoire

- Années 90 : téléphonie cellulaire, Programmation propriétaire environnements totalement fermés
- Parallèlement apparition des PDA (Psion, Apple, Palm, Microsoft…)
- Fin des années 90, évolution majeure de Java :version 1.2 rebaptisée un temps version 2.0. Décliné en 3 grandes "éditions" :
  - JEE : programmation des serveurs d'applications
  - JSE : programmation des "clients lourds"
  - JME : informatique embarquée

Java



- La technologie J2ME se compose d'une machine virtuelle et d'un jeu d'APIs appropriées pour fournir des environnements d'exécution sur mesure aux terminaux mobiles.

Java 2 Micro Edition puis JME

- Cette édition regroupe en fait différentes incarnations de Java différenciées par la notion de configuration et de profil :
  - Java Card orienté vers les cartes à puces
  - CLDC/MIDP : destiné à programmer des applications Java pour terminaux mobiles et PDA
  - CDC : supporté par les terminaux “haut de gamme” de l’époque et “set top box”, et autres “gateways”
  - DOJA : profil spécifique porté par NTT-DOCOMO (en France c’était BT avec i-Mode)

J2ME

- L'architecture se découpe en plusieurs couches :
  - Les configurations : définissent une plate-forme minimale en terme de services, comprenant : (i) une spécification de machine virtuelle dédiée (ii) un ensemble de classes de bases
    - CLDC (Connected Limited Device Configuration)
    - CDC (Connected Device Configuration).
  - Les profiles : spécification des caractéristiques communes d'utilisation (affichage, événements d'entrées/sorties : pointage, clavier, …). Les mécanismes de persistance (base de données légère intégrée)

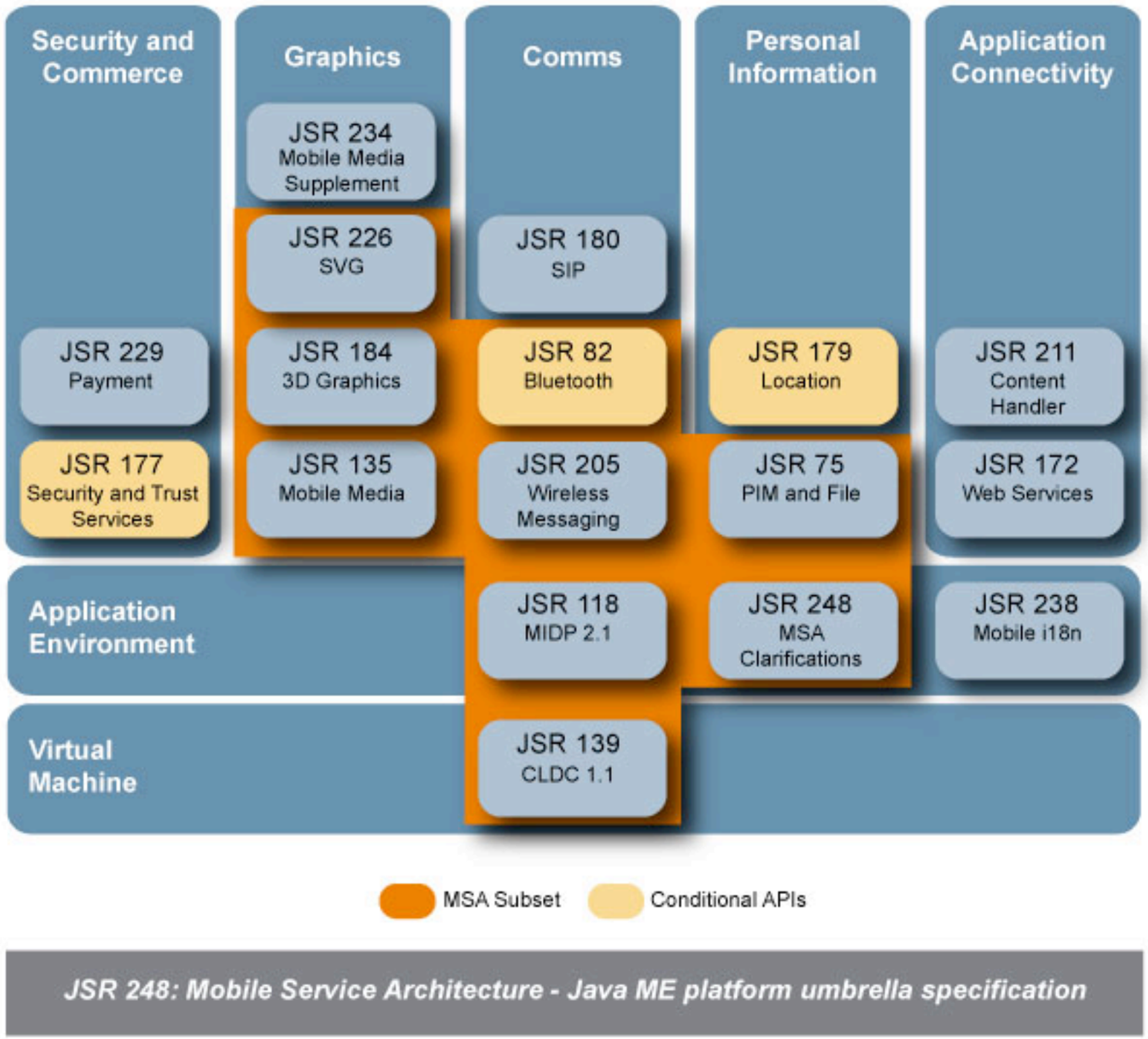
CDC, Fondation Profile, …

- CDC : Connected Device Configuration
  - gérer une grande variété d'IHM, voir pas d'IHM graphique
  - mémoire allouée de 2 à 16 Mo
  - processeur 16 bits ou 32 bits
  - connexion réseau
- Cible : set-top boxes, Consoles internet, routeurs et équipements réseau, PDA haut de gamme.
- Le Fondation Profile sert de brique de base et se focalise sur la gestion du réseau. Les profiles au dessus ajoutent des services (Personnal Profile pour l'interface graphique, RMI pour le support de Java-RMI…)

CLDC/MIDP

- Popularise l'idée de développer des applications sur téléphone mobile
- Le profile le plus déployé, supporté par la plupart des opérateurs
- Vocation principale : la programmation de "casual games"
- Définit une notion particulière d'application : la midlet, associée à un cycle de vie particulier et un modèle de sécurité

Les profiles sur CLDC



CLDC

- Minimum 160 à 500 Ko de (RAM+ROM)
- Processeur à 16-32 bits (vitesse 16Mhz ou +)
- Alimentation limitée, prise en charge d'une batterie
- Connexion au réseau (non permanente)
- Interface graphique limitée ou inexistante (en général, définie dans les profils)
- KVM
- Pré-vérification statique du bytecode +vérification à l'exécution (au chargement)
- Gestion adaptée des classes (jar+jad)

CLDC-1.0

- Pas de support pour les opérations flottantes
- Pas de gestion des “finalize” d’objets
- Nombre restreint d’erreurs (Exceptions)
- Pas de JNI (méthodes natives)
- Pas de ClassLoader utilisateur
- Pas d’introspection -> pas de RMI, de serialize…
- Pas de groupe de threads ou de demons
- Pas de Weak-References
- Pas de méthodes finalize

CLDC

- Bibliothèques principales :
  - java.lang.*
  - java.util.*
  - java.io.*
  - javax.microedition.io
- Modèle de sécurité
- Opérations d’entrée/sorties
- Support du réseau
- Internationalisation

Mécanismes et classes de base

- Le package java.lang :
  - Les interfaces : Runnable
  - Les classes : Boolean, Byte, Character, Class, Integer, Long, Math, Object, Runtime, Short, String, StringBuffer, System, Thread, Throwable
- On retrouve des services de bases de Java mais la plupart des classes ont des méthodes en moins ou une implantation simplifiée des certaines autres.
- Pas de gestion de processus externes, pas de clonage d'objet, pas de sérialisation, introspection très limitée, gestion simplifiée des propriétés de l'environnement

Mécanismes et classes de base

- Le package java.io :
  - Les interfaces : DataInput, DataOutput
  - Les classes : ByteArrayInputStream, ByteArrayOutputStream, DataInputStream, DataOutputStream, InputStream, InputStreamReader, OutputStream, OutputStreamWriter, PrintStream, Reader, Writer
- Mécanisme de base de gestion des flux de données de Java (mécanisme de gestion des entrées/sorties)

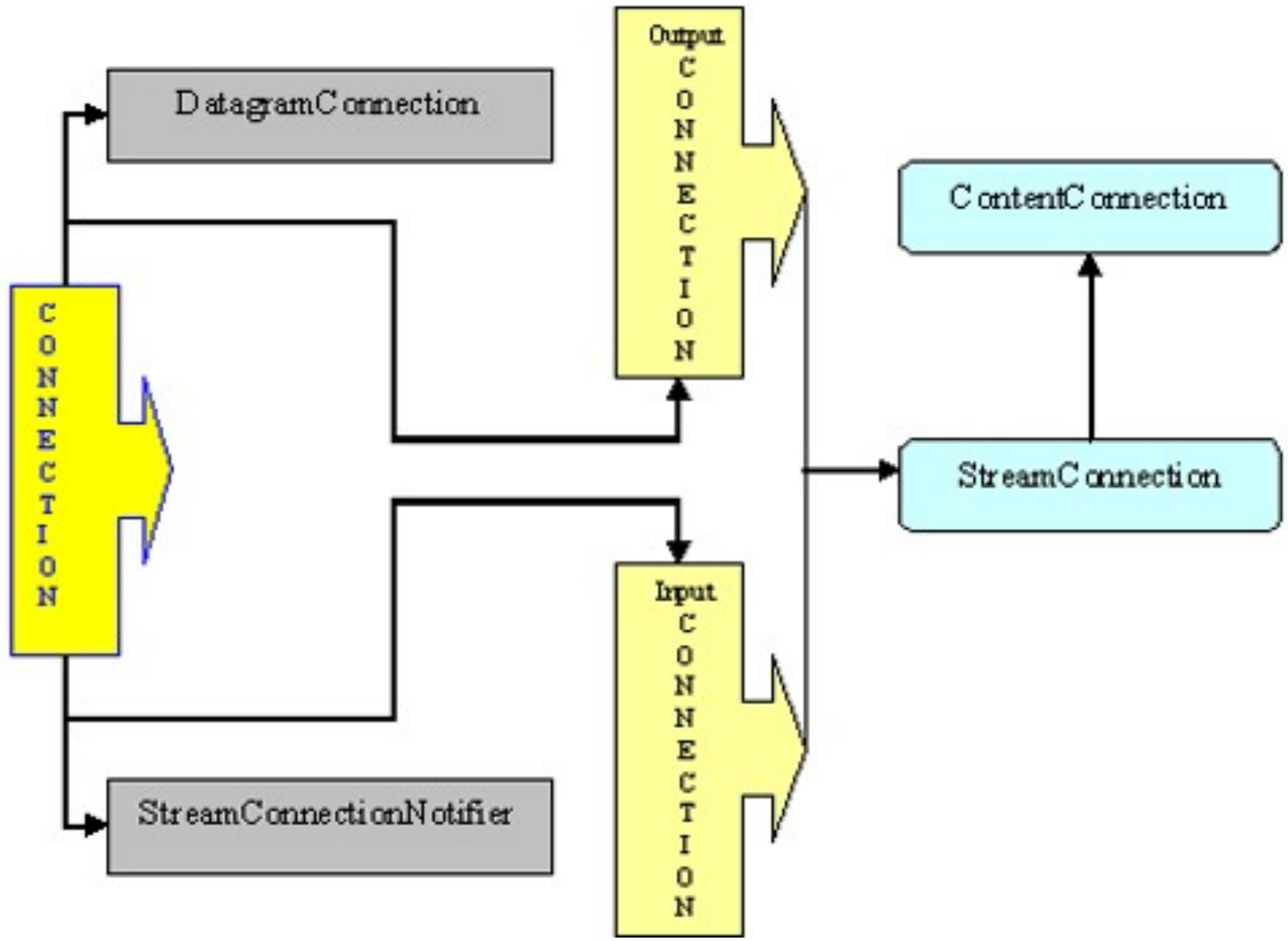
Mécanismes et classes de base

- Le package java.util :
  - Les interfaces : Enumeration
  - Les classes : Calendar, Date, Hashtable, Random, Stack, Timer, TimerTask, TimeZone, Vector
  - Gestion des dates et du “temps réel” (vient étendre le modèle de concurrence à base de threads par des timers)
  - Collections primitives d’objets à l’aide de classique Pile, table de hashage et tableau dynamique. Les itérateurs ne sont pas proposé mais on dispose cependant d’énumérations du contenu des collections

Mécanismes et classes de base

- Le package javax.microedition.io :
  - Les interfaces : Connection, ContentConnection, Datagram, DatagramConnection, InputConnection, OutputConnection, StreamConnection, StreamConnectionNotifier
  - La classe : Connector
- Défini le Generic Connection FrameWork (GCF) adaptation de Java au système mobile à capacité de connexion limité.
- Gère les objets de connections en particulier les connexions réseau ; aucune spécification de service à ce niveau

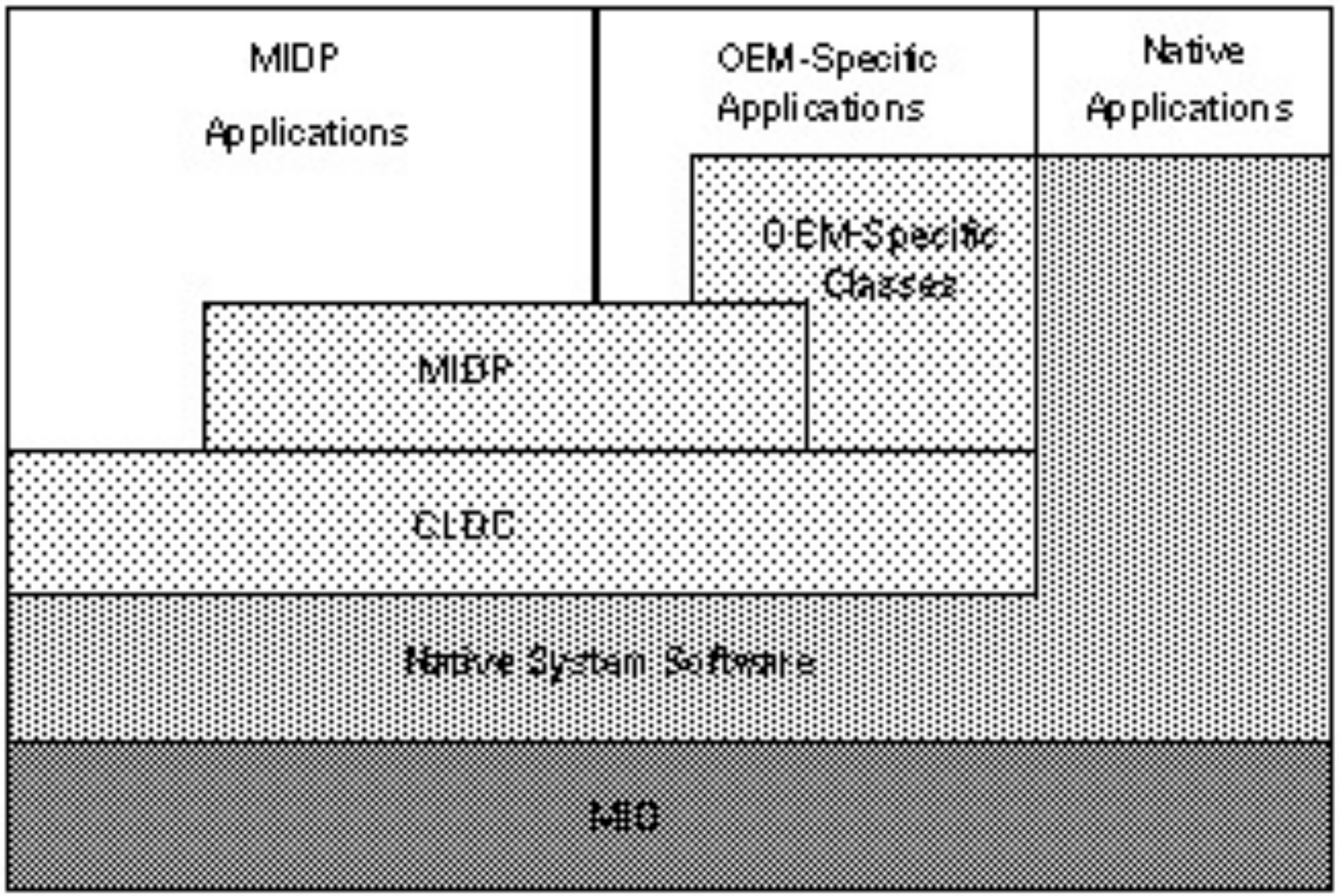
Mécanismes et classes de base



CLDC 1.1

- Support limité des Weak References (java.lang.ref)
- Ajout des calculs en flottant (Double, Float) support matériel recommandé
- Ajout Calendar, Date et TimeZone
- Minimum mémoire 160 -> 192 (principalement pour le support des opérations flottantes)
- Support des noms de threads
- Introduction de la méthode interrupt sur la classe Thread pour permettre de jouer plus finement avec l'ordonnancement

MIDP



Mécanismes et classes de base

- Le paramètre String de la méthode `Connector.open(String URL)` a le format suivant : *protocole:adresse:paramètres*. Exemples :
  - Connexion HTTP :

```
Connector.open("http://java.sun.com/developper?exemple=toto");
```

  - Connexion Socket :

```
Connector.open("socket://129.144.111.222:9000");
```

  - Connexion Datagram :

```
Connector.open("datagram://adress:port");
```

  - Communication à travers un port série :

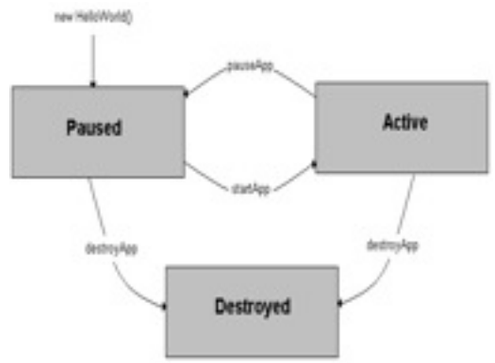
```
Connector.open("comm:com0;baudrate=9600");
```

  - Ouverture d'un fichier

```
Connector.open("file://myFile.txt");
```

MIDP

- Ciblé pour les téléphones mobiles/PDA : une connexion réseau pas si intermittente, un écran pas terrible
- L'application s'appelle la Midlet. 3 états :
  - Active (startApp)
  - Suspendue (pauseApp)
  - Détruite (destroyApp)



- RMS gestion d'enregistrements associés aux Mid-lets
- MIDP impose au minimum HttpConnection : com bidirectionnelle sans fil (non permanente), Datagram optionnel

MIDP

- Version 1 :
  - 128 Ko pour MIDP persistant
  - 8 Ko pour stocker le data de la Midlet persistant
  - 32 Ko pour le fonctionnement du Runtime Java (le tas et la pile)
- Version 2 :
  - 256 Ko persistant
  - 8 Ko app persistant
  - 128 Ko heap

MIDP

- Caractéristiques affichage min :
  - 96x54 pixels
  - 1 bit de profondeur
  - ratio 1:1
- Caractéristiques dispositifs d'entrées :
  - QWERTY kbd
  - ITU-T keyboard (téléphone)
  - touch pad

MIDP

- 3 packages :
  - javax.microedition.midlet (~Applet) : Socle technique destiné à gérer le cycle de vie d'une application
  - javax.microedition.lcdui : gestion de l'interface homme/machine (IHM).
  - Javax.microedition.rms : base de données persistante légère.

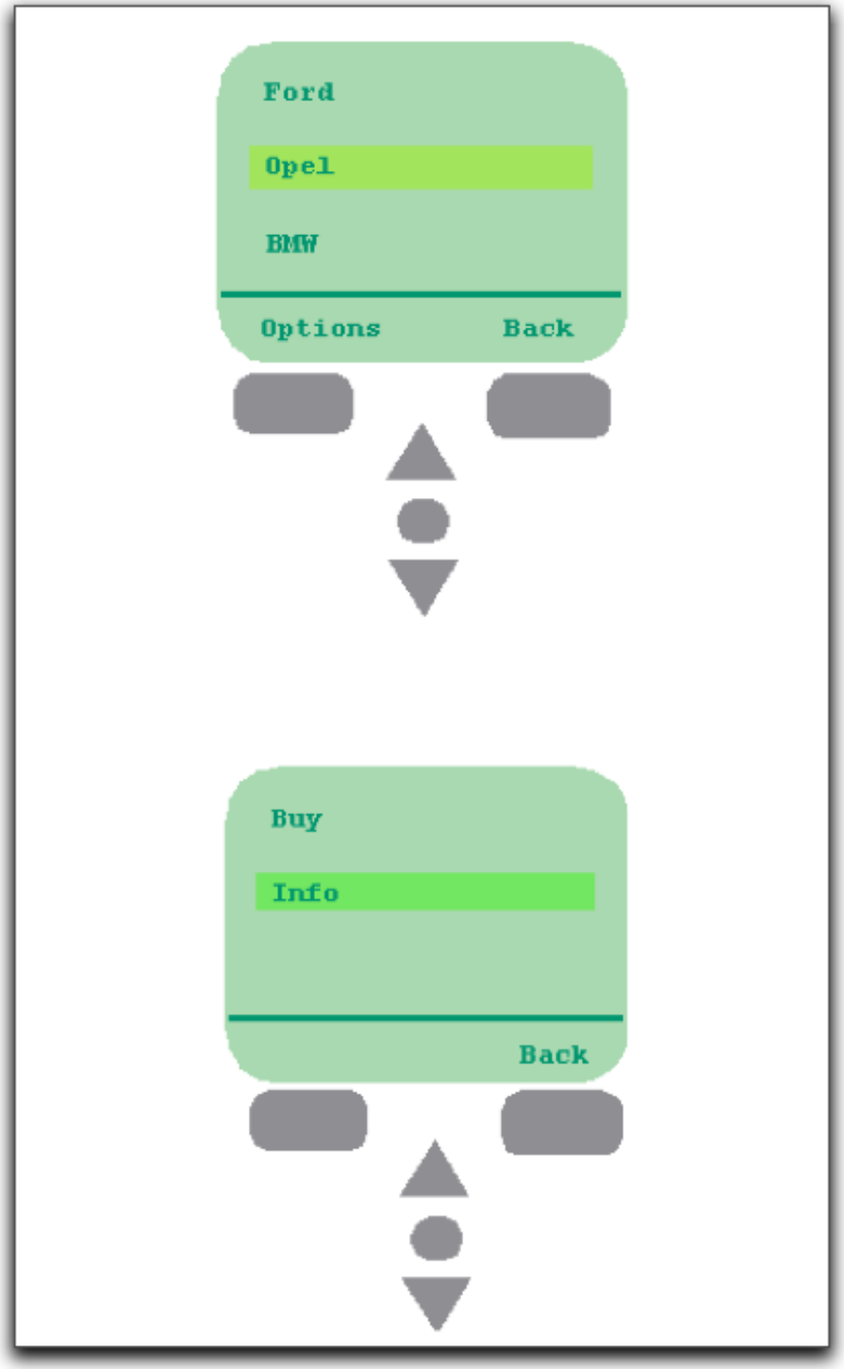
MIDP

- Une midlet active a accès à la totalité de l'écran, représenté par la classe l'unique instance de la classe `Display. (Display.getDisplay(myMidlet))`
- La midlet affiche à tout moment sur l'écran un objet `Displayable` : de type `Canvas` ou `Screen`
- 2 modes d'affichage :
  - une API de haut niveau : fournit des composants simples qui favorise la portabilité
  - une API de bas niveau : permet d'exploiter les fonctionnalités du terminal ; accès direct à l'écran et aux événements touches et système de pointage

MIDP

- Gestion de la persistance (RMS : Record Management Software)
  - Classe : RecordStore
  - API indépendante des terminaux
  - Un enregistrement est un tableau de bytes
  - sauvegarde possible en mémoire permanante
  - partage entre applications possible
  - Support pour les énumérations et les ensembles
  - Mise à jour atomique des enregistrements simples

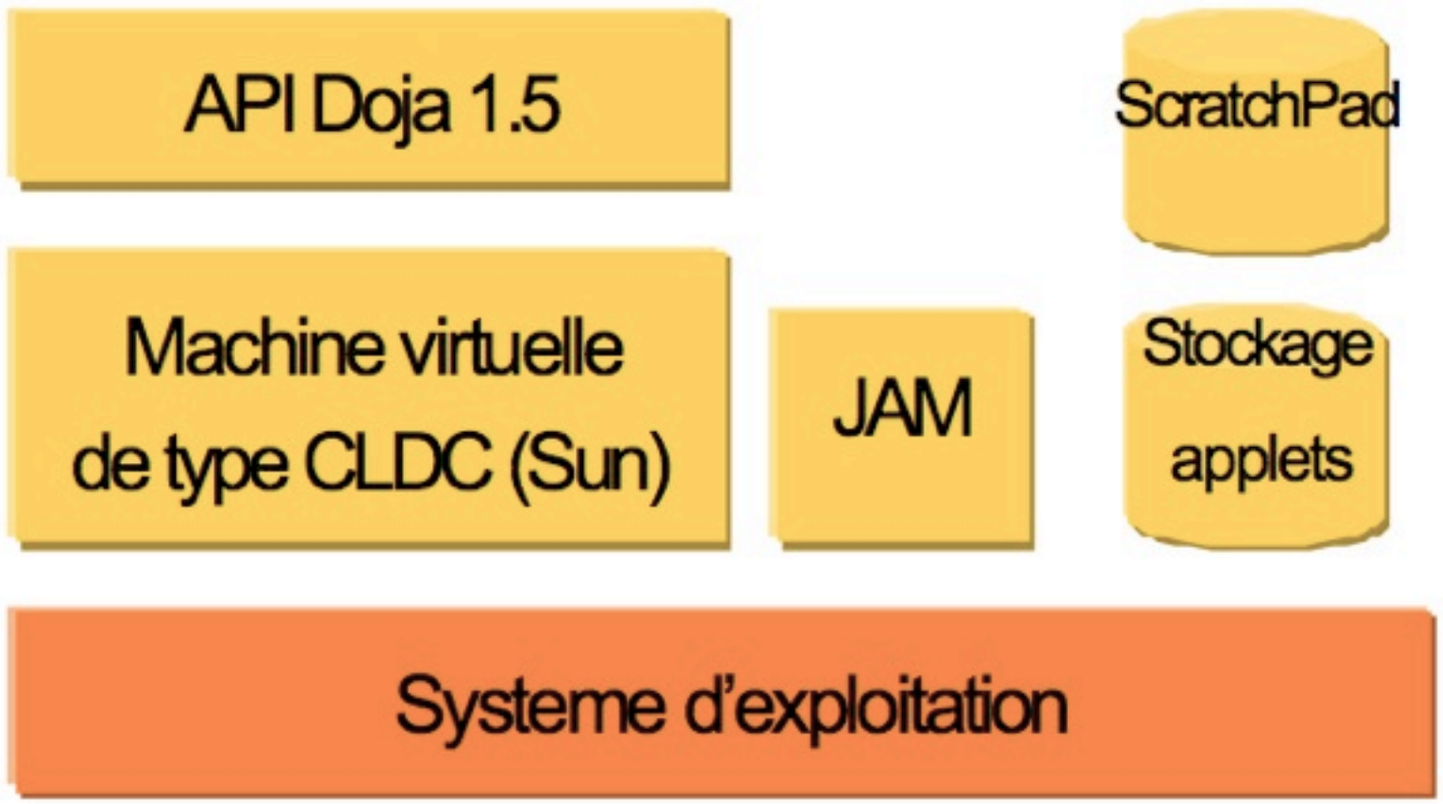
MIDP



DOJA

- Concurrent de MIDP développé par NTT DOCOMO au dessus de CLDC intégré à la solution i-Mode (HTML adapté au mobile)
- Spécifications plus précise de l'architecture matérielle et des services nécessaires. On retrouve les caractéristiques suivantes :
  - Support du HTTP, HTTPS
  - Composants génériques pour une interface de haut niveau et contrôle de bas niveau du graphisme
  - Zone de stockage des données (ScratchPad)
  - Téléchargement, sécurité, gestion des applis (JAM, ADF, …)

DOJA



2. Environnement applicatif Java pour i-mode

- com.nttdocomo.lang
- com.nttdocomo.io
- com.nttdocomo.net
- com.nttdocomo.ui

CE, Symbian, Palm, Brew, BlackBerry

- Convergence PDA et téléphone mobile
  - Les OS pour smart-devices proposent leurs propres solutions pour le développement d'applications (SDK, déploiement des applications -signatures-)
  - Développement assez complexes.
  - Les interfaces sont perçues comme des interfaces "Desktop" adaptées aux mobiles (clavier minimal, stylet, molette de défilement,…)
  - Réservé longtemps aux devices haut de gamme

➡ orientation vers les milieux professionnels

l'iPhone d'Apple

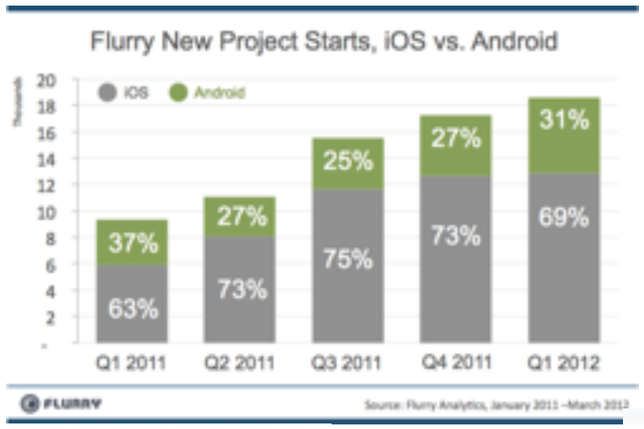
- Un démarrage en 2 temps :
  - juin 2007 : iPhone première génération bouleverse les usages et les standards d'IHM. Apple promet un kit de dev basé sur les technos Web (HTML 5)
  - juin 2008 : IOS 2, apparition de l'AppStore. Un SDK et un modèle économique très efficace
  - Fixe certains standards des smart-phones : Grand écran tactile multipoints, GPS, accéléromètres, compas, caméra,WiFi, GPRS (connectedless ?)
  - Transforme profondément la conception des applications

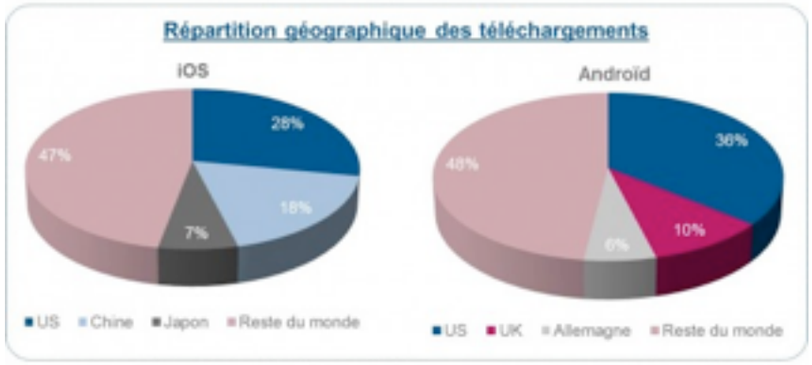
Android

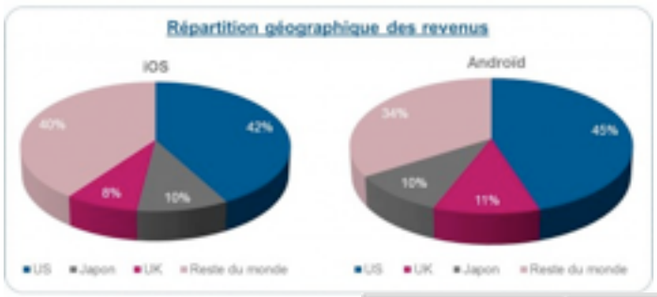
- 1 an plus tard Google embraye le pas d'Apple et lance Android
- Système ouvert, Open Source (licence Apache) basé sur un noyau Linux et sur le langage Java. Il reprend les grands principes d'iOS, et offre un environnement très complet à tous les constructeurs
- Le SDK est disponible pour un grand nombre de plate-forme, le langage de prédilection est Java afin de favoriser l'adoption par les développeurs. Possibilité de développer en Natif (C/C++) grâce au NDK, mais peu de support de déploiement

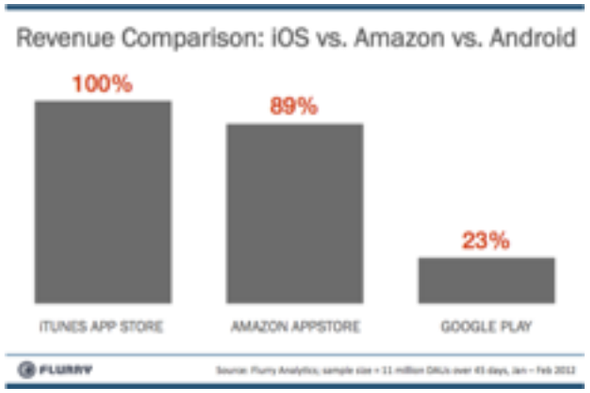
L'avenir ?

- Évolution extrêmement rapide, difficile à prédire (voir impossible)

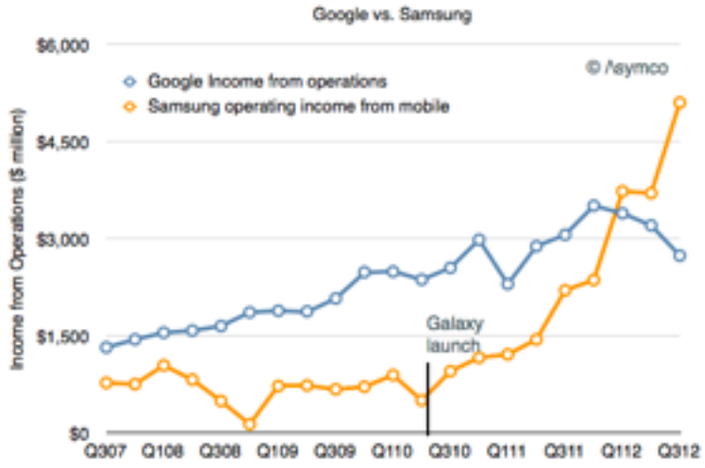








Operating System	3Q12 Shipment Volumes	3Q12 Market Share	3Q11 Shipment Volumes	3Q11 Market Share	Year-Over-Year Change
Android	136.0	75.0%	71.0	57.5%	91.5%
iOS	26.9	14.9%	17.1	13.8%	57.3%
BlackBerry	7.7	4.3%	11.8	9.5%	-34.7%
Symbian	4.1	2.3%	18.1	14.6%	-77.3%
Windows Phone 7/ Windows Mobile	3.6	2.0%	1.5	1.2%	140.0%
Linux	2.8	1.5%	4.1	3.3%	-31.7%
Others	0.0	0.0%	0.1	0.1%	-100.0%
Totals	181.1	100.0%	123.7	100.0%	46.4%



Les services de distributions d'application

- AppStore d'Apple : sert souvent de référence
- GooglePlay (ex Google Market) : en forte croissance, porté par la déferlante de nouveaux terminaux Android
- Windows Phone Store (ex MarketPlace) : assez loin derrière ses concurrents, mais Windows 8 changera-t-il la donne ?
- Amazon Appstore for Android : encore très jeune mais qui semble très rentable

➡ On peut difficilement ignorer ce "nouveau" terrain de jeu pour les développeurs