

# Projet de Fin d'Etude

# Les JAVA CARD



Année 2007-2008

Tuteur : Michel DUFNER

## **Résumé**

Le thème du projet est sur la technologie JavaCard. Cette technologie consiste au développement et à la mise en place d'applet sur des carte à puces intelligentes, les « Smart Cards ». Ces applets sont développées avec une sous partie du langage Java. Vous trouverez tout d'abord un état de l'art de la technologie ainsi qu'une présentation de tous les aspects qui sont nécessaires pour comprendre et développer des applications pour cette technologie. Ensuite je développerai un tutoriel expliquant comment installer sur son ordinateur un environnement de travail pour développer ces applications avec le matériel fourni. Ce tutoriel expliquera également comment prendre en main les logiciels nécessaires. Enfin la dernière partie traitera d'un exemple d'application pour comprendre concrètement le code d'une telle application.

## **Abstract**

The theme of the project is on the JavaCard technology. This technology consists with the development and the establishment of applet on smart card chips, « Smart Cards ». These applets are developed with a sub part of the Java language. You will find first a state of the art of this technology and a presentation of all the aspects that are needed to understand and develop applications for this technology. Then I develop a tutorial explaining how to install on your computer a work environment to develop these applications with the equipment provided. This tutorial also explains how to take control of the necessary software. The last part will deal with an example of a practical application to understanding the code of such an application.

# SOMMAIRE

Introduction	1
I. La technologie Java Card	2
Historique – Architecture	2
Le langage Java pour les Smart Cards	3
La Java Card Virtual Machine	4
L'environnement d'exécution Java Card	6
L'interface de programmation Java Card	7
Communications avec l'environnement extérieur	8
Les cycles de vie	11
Java Card : présent et avenir	12
II. L'environnement de développement	13
Installation de l'environnement	13
Prise en main du plug-in JCOP	17
Le Java Card Shell	20
III. Exemple d'application	23
Description du code	23
Tests avec le JCShell	27
Conclusion	31
Bibliographie	32
Annexes	33

## **Introduction**

Ce projet a pour but de me familiariser avec une technologie qui a émergé à la fin des années 90, qui est encore peu connue dans ses détails, bien que très utilisée. Cette technologie est celle des JavaCard, elle permet de développer en Java des applications pour des cartes à puce « Smart Card ».

Le premier des objectifs de ce projet a donc été de définir l'état de l'art de la technologie et surtout d'en comprendre le fonctionnement. Je présenterai donc dans un premier temps le fruit de mes recherches sur le sujet en expliquant l'ensemble des spécificités de la technologie.

Le but principal de ce projet est la prise en main de la technologie pour arriver à développer des applications. Pour cela, l'école a fourni le matériel nécessaire pour disposer d'un environnement de travail adéquat. Je détaillerai donc un chapitre explicatif pour apprendre à se servir de ce matériel.

Enfin je décrirai un exemple d'application pour comprendre concrètement le fonctionnement d'un applet Java Card, au travers de son code source.

# **I – La technologie Java Card**

## **Historique**

C'est en 1996 qu'un groupe d'ingénieurs Texan de Schlumberger ont voulu simplifier la programmation des cartes à puces tout en préservant la sécurité des données. C'est finalement le langage Java qui a été retenu. C'est cependant un sous-ensemble de Java qui a été utilisé à cause de la faible quantité de mémoire disponible.

Ainsi fut créé Java Card, le premier langage orienté objet pour carte à puce. Schlumberger (devenu Axalto) et Gemplus (tous deux réunis plus tard dans Gemalto) sont les cofondateurs du JavaCard Forum (<http://www.javacardforum.org>) qui regroupe les fabricants de silicium, les encarteurs et les clients, et qui recommande des spécifications à Sun Microsystems en vue d'obtenir une standardisation du langage.

## **Architecture**

Les Java Card sont des cartes à puce du type *Smart Card*. Ce sont des cartes à microprocesseur qui peuvent gérer des données sur 8, 16 ou 32 bits. Ce type de gestion est dû au fait que la carte fonctionne sur un principe de décodage d'octet. Voici les spécifications qui sont aujourd'hui les plus utilisées pour ce type de carte :

### **CPU :**

Microcontrôleur 8 à 32 bits :

### **Mémoire :**

ROM : 256 – 512 ko (48ko au minimum)

RAM : 512 ko (2ko au minimum)

EEPROM ou FLASH : 128 – 256 ko (32ko au minimum)

## Le langage Java pour les Smart Cards

L'utilisation de Java apporte plusieurs avantages par rapport à d'autres langages informatiques. Tout d'abord c'est un langage de haut niveau, orienté objet, qui permet de développer des applications de façon plus aisée. Même si les Java Card communiquent avec un système de byte code, il est facile de manier ce type de communication avec Java.

Ensuite l'utilisation d'une machine virtuelle permet aux applications embarquées de rester complètement indépendante du hardware, de l'OS et des éventuelles spécifications du fabricant de la carte.

Enfin Java apporte toutes ses propriétés en terme de sécurité des données. (cryptographie, authentification, confidentialité...)

A cause de la quantité de mémoire limitée le langage Java pour les Smart Card a été considérablement allégé et réduit au strict nécessaire pour effectuer ses propres applications.

### Les types de bases :

Supportés : boolean, bytes, short et int.

Non supportés : long, float, double, char, classe String

### Les tableaux :

Seuls les tableaux à une dimension sont autorisés, uniquement avec les types de bases cités plus haut.

### Programmation orientée objet :

Les objets de bases sont présents :

java.lang.Object

java.lang.Throwable

Le mécanisme d'héritage est semblable à celui de Java. On utilise les mots clés *super*, *instanceof*, *this*.

Il est possible d'utiliser des méthodes abstraites et des interfaces.

### Les exceptions :

Les classes Throwable, Error et Exception sont supportées. Comme pour Java, les exceptions peuvent être renvoyée (throw) ou attrapée (catch).

### D'autres absences notables dans Java Card :

- Pas de ramasse miette (Garbage Collector)
- Pas de classe Thread : pas de synchronisation
- Pas de sérialisation d'objet

## **La Java Card Virtual Machine**

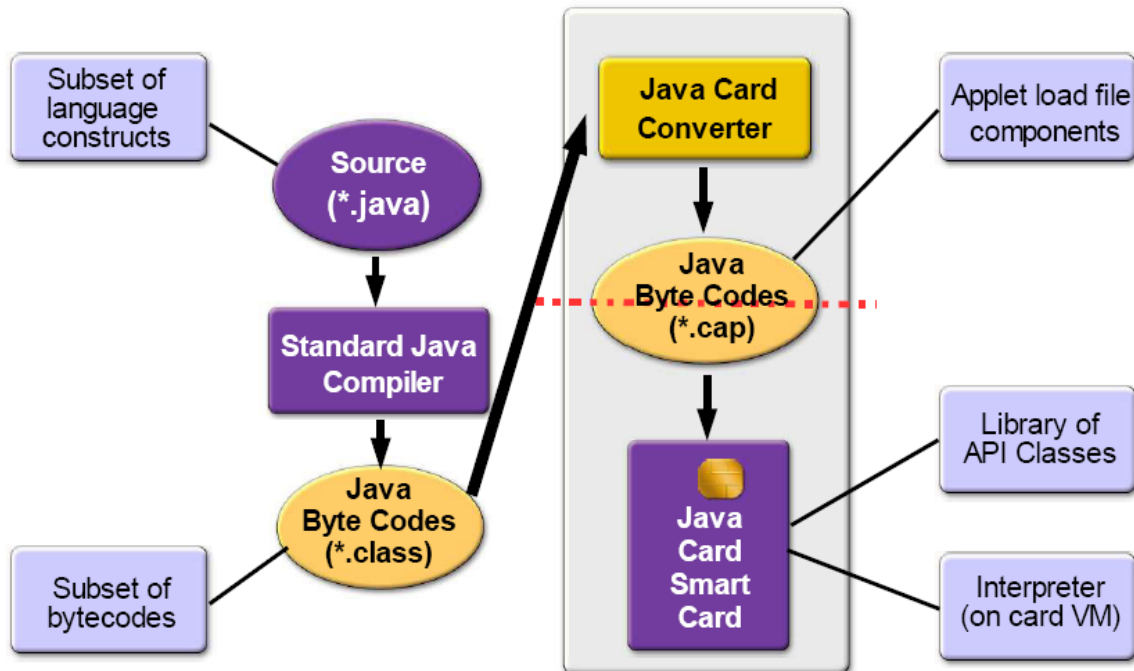
La JCVM (JavaCard Virtual Machine) est composé en deux parties : le convertisseur, situé sur l'ordinateur, et l'interpréteur de byte code, situé dans la JavaCard.

Le convertisseur :

Contrairement à la JVM qui ne traite qu'une classe à la fois, l'unité de conversion du convertisseur est un package. Les fichiers *.class* sont produits par le compilateur java à partir du code source. Ensuite, le convertisseur prétraite tous les fichiers *.class* contenu dans un package et les convertit en un fichier CAP (Converted Applet). Le fichier CAP est ensuite chargé sur une Java Card et exécuté par l'interpréteur. Durant le processus de conversion, le convertisseur exécute des tâches qui sont normalement exécutées par la JVM au moment du chargement des classes :

- Il vérifie que les images des java classes sont bien formées
- Il vérifie qu'il n'y a pas de violations du sous-langage java
- Il initialise les variables statiques
- Il convertit des références symboliques de classes, des méthodes et des champs en une forme plus compacte supportée par les Java Card.
- Il alloue, stocke et crée des structures de données de la machine virtuelle pour représenter les classes.

Le convertisseur ne prend pas seulement en entrée les fichiers *.class* à convertir en fichiers *.CAP* mais aussi un ou plusieurs fichiers d'exportation. Si le package importe des classes d'autres packages le convertisseur charge aussi les fichiers d'exportation de ces packages. Le convertisseur produit donc un fichier CAP et un fichier d'exportation du package converti.



L'interpréteur de byte code :

L'interpréteur, en fournissant un support d'exécution du langage Java, permet aux applets d'être indépendantes du matériel. Il exécute en particulier les tâches suivantes :

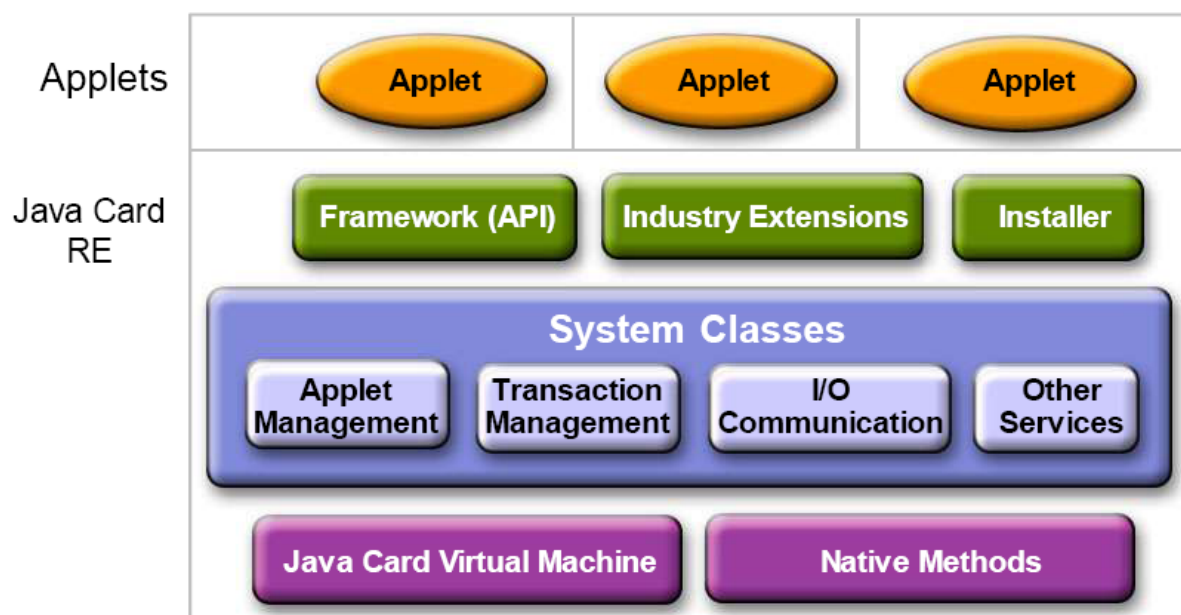
- Il exécute les instructions du bytecode et des applets
- Il contrôle l'allocation mémoire et la création d'objets
- Il sécurise le temps d'exécution

La machine virtuelle Java Card a été décrite comme comprenant le convertisseur et l'interpréteur. Elle est toutefois définie informellement comme étant la partie située sur la carte, à savoir l'interpréteur actuellement décrit. Ainsi concernant les Java Cards, on peut faire l'analogie entre l'interpréteur et la machine virtuelle, contrairement à la machine virtuelle Java qui elle, inclue le convertisseur et l'interpréteur.



## L'environnement d'exécution Java Card

L'environnement d'exécution Java Card (JCRE) gère les ressources de la carte, la communication avec le réseau, l'exécution des applets et leur sécurité. En fait c'est le système d'exploitation des smart cards. Le JCRE est situé au sommet de l'architecture d'une smart card. Il consiste en la machine virtuelle Java (l'interpréteur de bytecode), la structure des classes d'applications Java Cards (APIs), des extensions industrielles spécifiques, et les classes systèmes. Le JCRE sépare habilement les applets de la propriété de la technologie des smart cards des vendeurs. Les applets sont ainsi plus facile à écrire et sont indépendantes de l'architecture privée des smart cards.



La période pendant laquelle la carte est insérée dans une machine est appelée session CAD (Card Acceptance Device). Durant une session CAD, le JCRE opère comme une smart card en établissant une communication APDU (Application Protocol Data Unit) d'entrée-sortie avec un serveur. Les APDUs sont des paquets de données échangés entre des applets et une application hôte. Chaque APDU contient soit un ordre du client à l'applet, soit une requête de l'applet au client.

Après l'initialisation du JCRE, celle-ci entre dans une boucle, en attente d'un ordre APDU du client. Celui-ci envoie des ordres APDUs à la Java Card par l'intermédiaire d'une communication utilisant les points de contacts d'entrée-sortie. Quand une commande arrive, le

JCRE sélectionne soit une applet à exécuter comme indiqué dans la commande, soit envoie directement la commande à une applet déjà sélectionnée. L'applet sélectionnée prend alors le contrôle et exécute la commande APDU. Une fois la commande exécutée, l'applet envoie une réponse à l'application hôte et redonne le contrôle au JCRE. Le procédé se répète à l'identique quand une nouvelle commande arrive.

## **L'interface de programmation Java Card**

Les APIs Java Card consistent en un ensemble de classes spécialisées dans la programmation d'applications de smart card conformément à la norme ISO 7816.

Les APIs contiennent trois packages principaux et un package d'extension. Les trois packages principaux sont les packages `java.lang`, `javacard.framework`, et `javacard.security`. Le package d'extension est `javacardx.crypto`.

Les classes dans ces APIs sont compactes et succinctes. Elles incluent des classes adaptées à la plateforme java fournissant un support sur le langage et des services de cryptographie. Elles contiennent également des classes spécialement conçues pour supporter le standard ISO 7816 des smart cards.

### *Le package `java.lang`*

Ce package est un sous-ensemble du package java équivalent `java.lang`. Les classes supportées sont `Object`, `Throwable`, et quelques classes d'exceptions reliées à la machine virtuelle. Beaucoup de méthodes java ne sont toutefois pas disponibles dans ces classes. La classe Java Card `Object` par exemple ne définit qu'un constructeur par défaut et la méthode *equals*.

### *Le package `javacard.framework`*

Le package `javacard.framework` est essentiel. Il fournit l'architecture des classes et des interfaces pour le fonctionnement basique des applets Java Card. Plus important, il définit la classe de base `Applet` qui donne le cadre d'exécution et d'interaction des applets durant leur fonctionnement.

Deux autres classes importantes de ce package sont la classe APDU qui sert aux protocoles de transmission et la classe PIN qui gère les mots de passe.

#### *Le package javacard.security*

Ce package est utilisé pour les fonctions de cryptographie de la plate-forme Java Card. Son implémentation est basée sur le package Java du même nom java.security. Il définit notamment la classe keyBuilder et des interfaces variées représentant le système cryptographique de clés (DES) et (DSA et RSA).

#### *Le package javacardx.crypto*

Comme dit précédemment ce package est un package d'extension. Il contient des classes de cryptographie et des interfaces dont les spécifications sont régulées par les Etats-Unis. Il définit la classe abstraite Cipher pour l'encryption et la décryption des données.

## **Communications avec l'environnement extérieur**

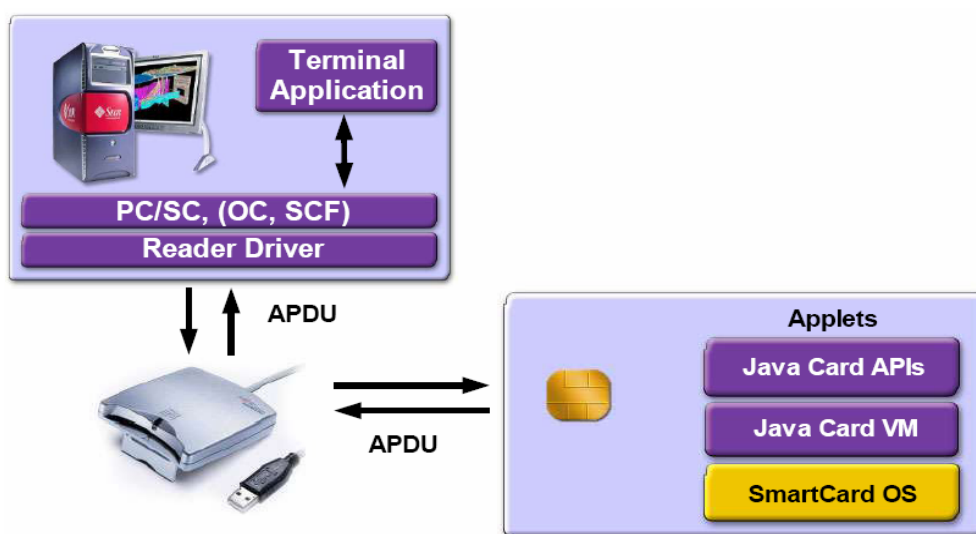
La Java Card est un module de traitement réactif à l'environnement. C'est une application externe qui va appeler les applets présents dans la carte. Cette application va envoyer des requêtes à la carte pour exécuter les applets qui lui sont nécessaires. Chaque applet présent dans une Java Card est spécifique du programme client. Chacun d'entre eux est ainsi numéroté avec un unique identifiant : l'AID (Application Identifier).

Dans la technologie Java Card, à chaque applet est assigné un AID. De même chaque paquetage Java est identifié par un AID. Ceci est la convention de nommage définie par la norme ISO 7816. Un AID est une séquence d'octets comprises entre 5 et 11 octets. Voici comment ils sont organisés :

National Registered Application Provider	Proprietary Application Identifier Extension
5 octets	0 – 11 octets

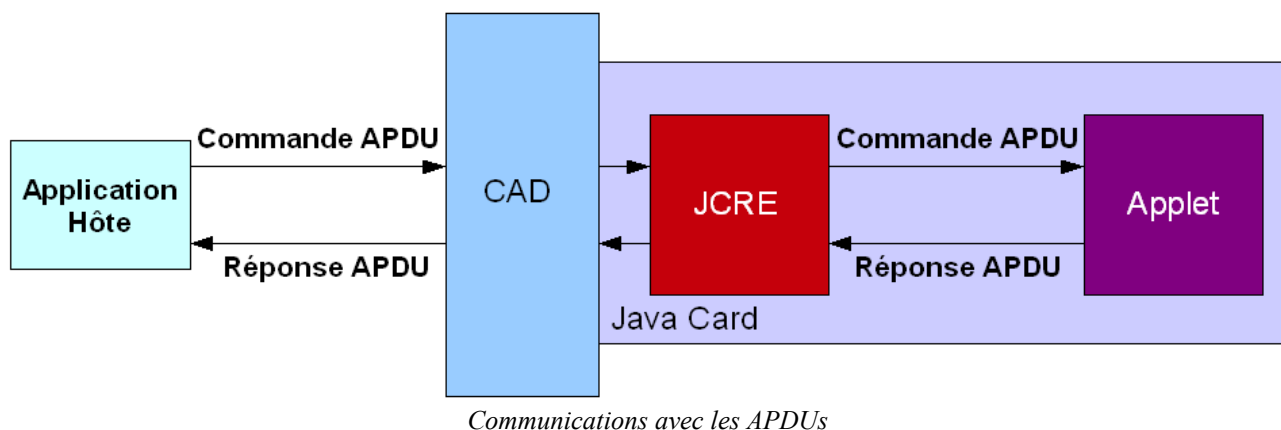
Le RID est fourni aux compagnies par l'ISO. Chaque compagnie a donc un RID unique. Ensuite ce sont ces compagnies qui complètent l'AID avec le PIX de leur choix. A l'arrivée, chaque application a donc un AID unique : [RID,PIX]

Les applets situés sur une Java Card communiquent avec l'application cliente grâce aux échanges de requêtes et de réponse, les APDUs.



Les communications APDU vont toujours par paire. Chaque paire étant composée d'un requête (ou commande) de l'application cliente vers la Java Card, et d'une réponse qui est le résultat de la requête. Les cartes n'initient jamais de commande APDU, elles sont toujours en attente d'une requête APDU.

L'application envoie une commande APDU via le lecteur (CAD). Le JCRE en recevant une commande, sélectionne une nouvelle applet ou bien passe la commande à une applet courante (déjà sélectionnée). L'applet courante traite la commande et retourne une réponse APDU à l'application. Les commandes et réponses APDU sont échangées alternativement par la carte et le CAD.



Format d'une commande APDU :

En-tête obligatoire				Corps optionnel		
CLA	INS	P1	P2	Lc	Données	Le

- CLA (1 octet) : Classe d'instructions : Indique la structure et le format pour une catégorie de commandes et de réponses APDU
- INS (1 octet) : code d'instruction : spécifie l'instruction de la commande
- P1 et P2 (1 octet chacun) : paramètres de l'instruction
- Lc (1 octet) : nombre d'octets présents dans le champ données de la commande
- Données (de taille « Lc » octets): une séquence d'octets de données
- Le (1 octet) : nombre maximum d'octets attendus dans le champ de données de la réponse APDU.

Format d'une réponse APDU :

Corps optionnel	Champs obligatoires	
Données	SW1	SW2

- Données : champs de données de longueur variable (« Le » octets au maximum)
- SW1 et SW2 (1 octet chacun) : « Status Word ». Octets d'état de traitement de la Java Card.

Tous les champs CLA, INS, P1, P2, SW1 et SW2 ne sont pas arbitraires. Les valeurs de ces champs sont définies suivant la norme ISO 7816. Suivant le type de requête, il existe un code prédéfini pour chaque champ. La liste de ces codes prédéfinis est présente en annexe.

## Les cycles de vie

### *Le cycle de vie de la carte :*

L'environnement d'exécution Java Card est initialisé à la phase d'initialisation de la carte. Le cycle de vie de la carte est donc le même que celui du JCRE. Quand la carte est hors tension, la machine virtuelle est suspendue, et non arrêtée. Les états du JCRE et des objets persistants sont donc préservés.

Une fois que la carte est à nouveau sous tension, le JCRE relance l'exécution de la machine virtuelle en chargeant les informations sauvegardées dans la mémoire persistante. Il existe des objets persistants et temporaires. Les persistants existent donc au travers des sessions CAD tandis que les temporaires ne sont pas sauvegardés à la fermeture de la session. Durant un reset, si une transaction n'a pas été terminée, le JCRE nettoie toutes les données nécessaires pour revenir à un état cohérent.

Chaque opération d'écriture de la machine virtuelle est atomique. Le champ traité prend sa nouvelle valeur ou est restauré à la valeur précédente. C'est ainsi que chaque transaction est bloc d'opérations atomiques.

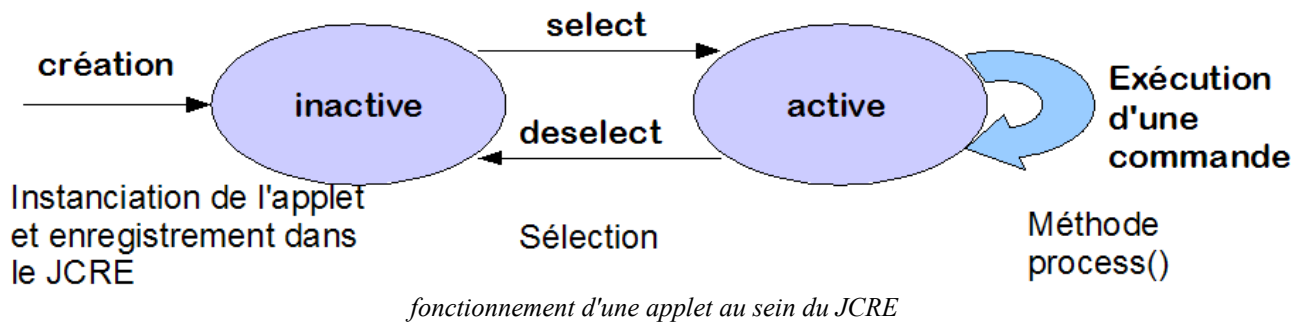
### *Le cycle de vie d'un applet :*

Une applet est une application serveur. Elle est sélectionnée à partir de son identifiant unique, l'AID. Cette sélection se fait grâce à une commande APDU au travers d'un terminal.

Chaque applet s'exécute dans un espace qui lui est propre. La JCVM doit assurer cette propriété. Une applet ne peut avoir d'effet sur une autre applet. Il existe des mécanismes de partage, qui permettent à un applet d'accéder à des services offerts par une autre applet ou par le JCRE. Cette caractéristique est le « firewall » des applets.

Une applet hérite de la classe `javacard.framework.Applet`. Pour interagir avec le JCRE, il doit implémenter les méthodes publiques suivantes :

- `select` / `deselect` : pour activer ou désactiver une applet
- `install` / `uninstall` : pour installer ou supprimer une applet sur la carte
- `process` : pour traiter les commandes APDU et envoyer une réponse APDU
- `register` : pour enregistrer l'applet auprès du JCRE.



## Java Card : présent et avenir

Aujourd'hui les Java Card sont largement répandues et utilisées à travers le monde. On estime leur nombre en circulation à plus d'un milliard. Les applications courantes qui les utilisent sont les cartes SIM 3G, les cartes d'identification d'entreprises ou gouvernementales, les cartes de paiement pour décodeur TV, la finance, etc. Ses cartes peuvent être utilisées avec ou sans contact (RFID).

La plate-forme Java Card développée par Sun Microsystems en est, depuis 2005, à la version 2.2.2. Une nouvelle génération de Java Card est prévue pour un avenir très proche (2008). En plus d'une augmentation des caractéristiques matérielles (augmentation de la mémoire et de l'horloge), il est prévu d'implémenter des classes supplémentaires comme `String`, `Hashtables` ainsi que le ramasse miette (`Garbage Collector`). De plus, ces nouvelles cartes pourraient reconnaître directement le protocole TCP/IP sans passer par un ordinateur intermédiaire qui traduirait les communications dans un format APDU standard. Il y aurait alors deux sortes d'applets, celles qui communiquent avec des trames APDU classiques, et d'autres dédiées à l'internet, qui communiqueraient avec un nouveau type d'APDU permettant d'échanger des trames IP.

## **II – L'environnement de développement**

### **Installation de l'environnement**

Pour pouvoir développer des applications Java Card dans le but de comprendre concrètement son fonctionnement ainsi que de réaliser des exemples d'applets, nous avons à notre disposition le matériel nous permettant de pratiquer la technologie Java Card :

- Un lecteur de carte Java Card de marque MO Techno (modèle ACSACR 38)
- Un jeu de plusieurs carte Java Card, dont une carte d'activation
- Un CD d'installation : driver + exemples d'utilisations

Pour utiliser tout le matériel fourni, MO Techno (Mobile Technologies) nous indique qu'il faut utiliser le logiciel Eclipse sous Windows, accompagné d'un plug-in particulier appelé *JCOP plug-in for Eclipse*. Ce plug-in, développé par *IBM* était jusque récemment en libre téléchargement sur internet. En plus de ce plug-in, toute la documentation détaillée était également consultable en ligne. En Juillet 2007, ce plug-in a été licencié par *NXP Semiconductors*, filiale de Philips. Depuis il est impossible de trouver ce plug-in ainsi que toute documentation officielle sur ce plug-in sans l'accord d'*IBM* et de *NXP*.

Heureusement l'an dernier un élève a travaillé sur le même sujet avec ce même matériel. A l'époque il avait pu récupérer ce plug-in. Grâce à mon tuteur j'ai pu le récupérer et l'installer avec Eclipse. Pour les personnes souhaitant travailler avec ce plug-in, je l'ai mis en téléchargement à cette adresse : <http://www.megaupload.com/fr/?d=HVKH3RAA>

Le CD fournit contient également un tutoriel d'installation en anglais et allemand. Je vais décrire à présent la marche à suivre pour installer le lecteur de carte et le plug-in JCOP.

Le lecteur de carte Java Card MO Techno se branche sur un port USB de l'ordinateur. On peut laisser Windows installer le driver automatiquement. Si cela ne marche pas, on peut utiliser le driver présent sur le CD. Pour cela, il faut décompresser l'archive nommée « ACR38U\_inst\_111000\_P » et exécuter le Setup obtenu après décompression.

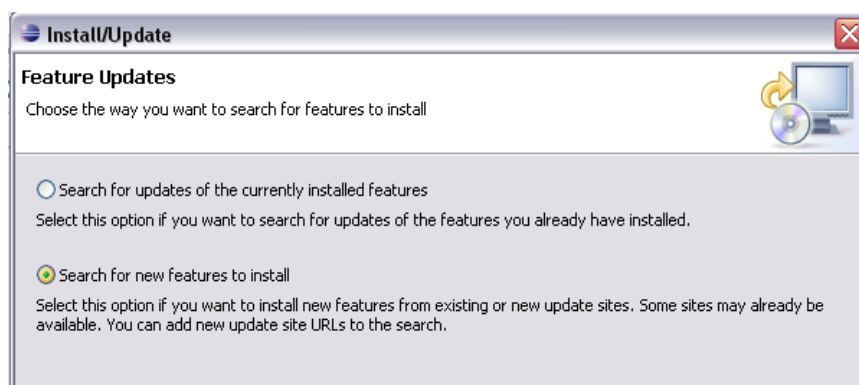
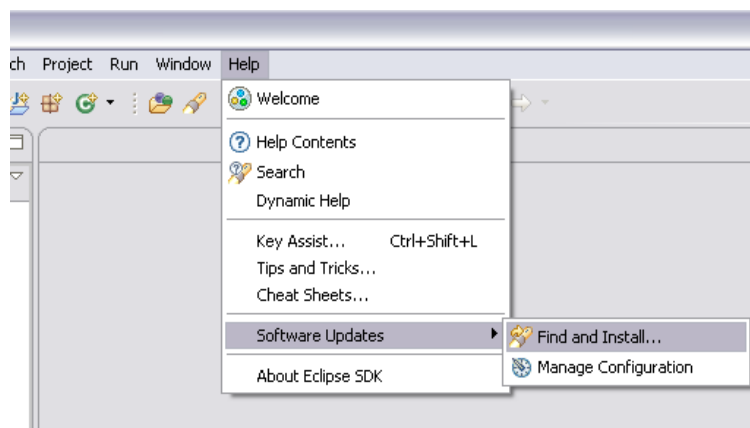


A ce stade, Windows devrait reconnaître correctement le lecteur de carte. Maintenant il faut installer l'environnement de développement Eclipse avec son plug-in dédié JCOP.

Avant toute chose il faut bien évidemment s'assurer que l'environnement Java est installé sur l'ordinateur. Vous pouvez télécharger la dernière version de la JRE (Java Runtime Environment) à cette adresse : <http://www.java.com/en/download/manual.jsp>

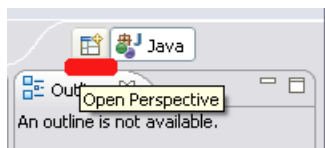
On trouve toutes les versions téléchargeables d'Eclipse à l'adresse suivante : <http://archive.eclipse.org/eclipse/downloads/>. Attention, il faut utiliser la version 3.2.2 pour Windows, la version 3.3 ne faisant pas fonctionner correctement le plug-in JCOP.

Lancez Eclipse et définissez le chemin de votre répertoire de travail (workspace). Pour installer le plug-in, procédez de la façon suivante :

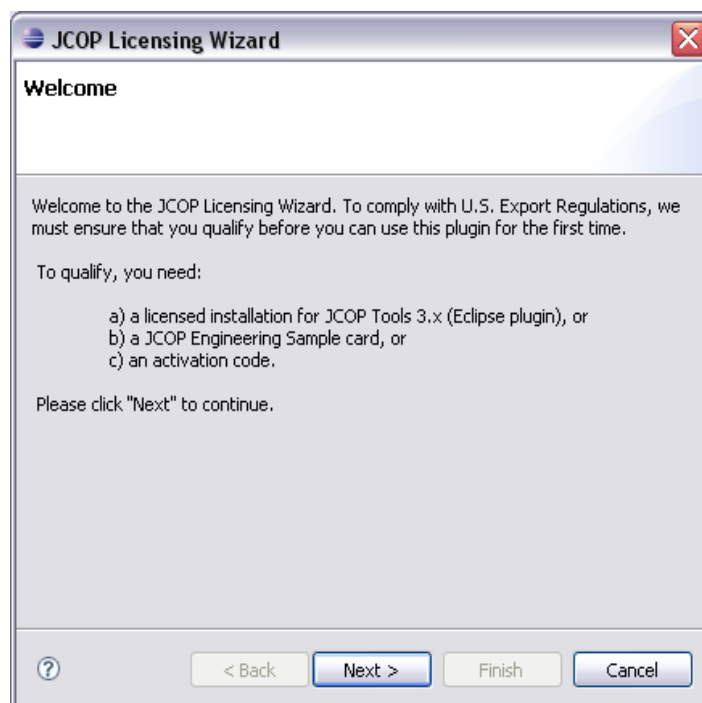


Effectuez les opérations décrites précédemment et cliquez sur le bouton « New Archived File » et sélectionnez le fichier « tools.zip ». Cliquez sur « OK » puis « Finish ». Laissez-vous guider pendant la suite de l'installation, acceptez la licence et installez tous les fichiers nécessaires. Enfin redémarrez Eclipse pour pouvoir utiliser le plug-in.

L'installation terminée, il faut maintenant activer le plug-in en utilisant la Carte de validation fournie. Branchez le lecteur de carte sur l'ordinateur et insérez cette carte dans le lecteur. Pour travailler avec ce plug-in, il faut sélectionner la perspective qui lui est dédiée dans Eclipse. Pour cela, cliquez sur le bouton jaune en haut à gauche (ci-dessous souligné en rouge) :



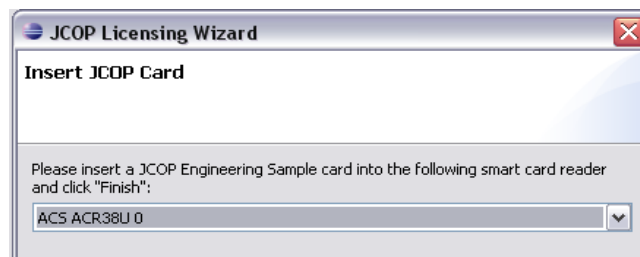
Pour accéder à toutes les perspectives, cliquez sur « Other » et sélectionnez la perspective « JCOP Debug ». Comme c'est la première fois que l'on utilise le plug-in, le protocole de validation du plug-in se met en route. Cliquez sur « Next » :



Il y a trois façons possibles d'activer le plug-in. Nous allons utiliser la vérification par une carte d'authentification. Assurez-vous que la carte « Engineering Sample Card » est dans le lecteur :



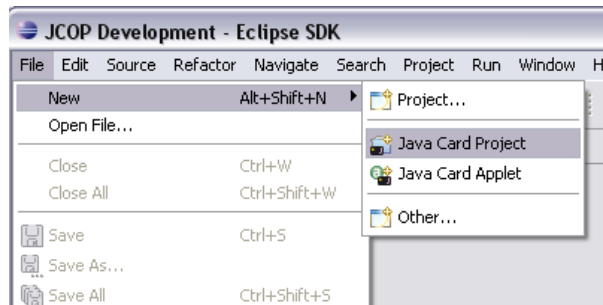
Indiquez quel lecteur vous souhaitez utiliser pour lire la carte d'authentification. Sélectionnez le lecteur MO Techno ACS ACR38U :



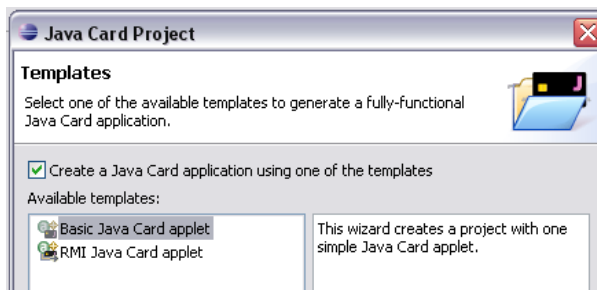
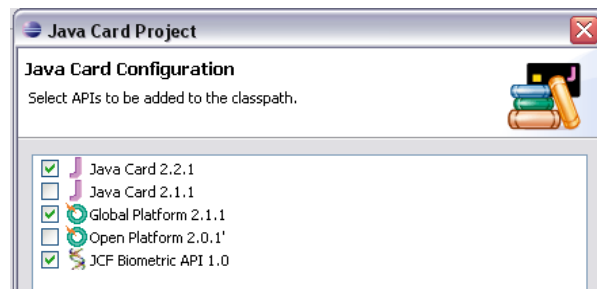
Finissez la démarche d'activation. Vous pouvez maintenant utiliser le plug-in JCOP pour développer des applets pour Java Card.

## Prise en main du plug-in JCOP

Nous pouvons maintenant créer un nouveau projet Java Card :

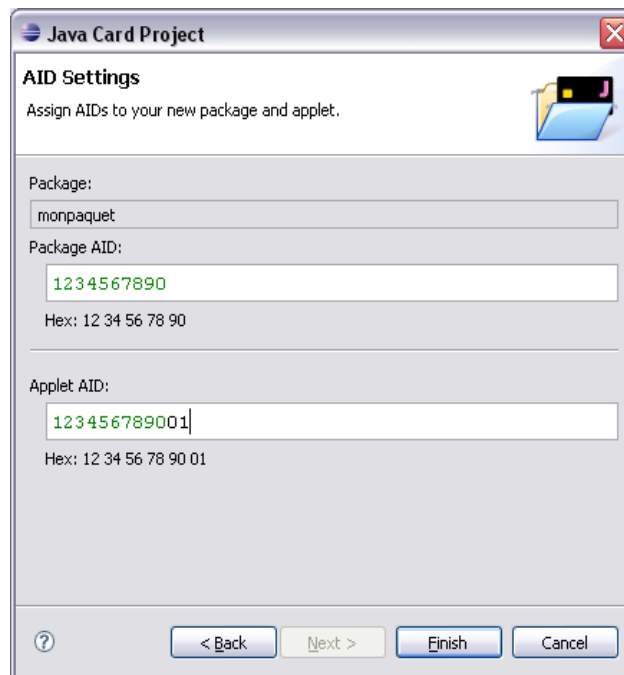


On donne le nom du projet et on choisit les différentes options possibles :



On peut choisir entre autre la version de l'implémentation Java Card et si notre applet utilisera l'implémentation RMI pour Java Card.

On entre le nom du package et de notre applet pour ensuite donner à chacun d'entre eux son AID :

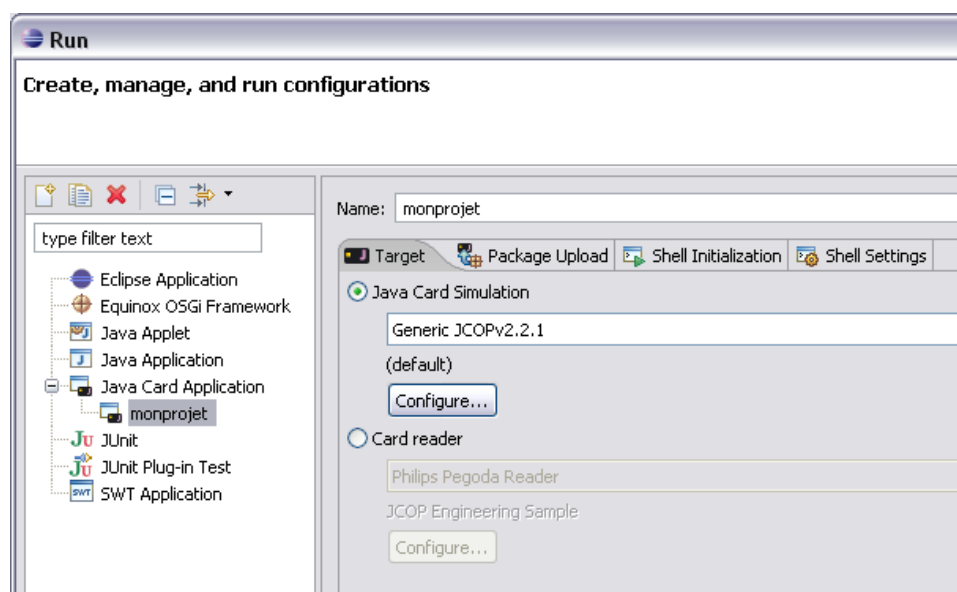


The screenshot shows a dialog box titled "Java Card Project" with a sub-header "AID Settings". Below the sub-header is the instruction "Assign AIDs to your new package and applet." and a small icon of a Java card. The dialog is divided into two sections. The first section, labeled "Package:", contains a text field with the value "monpaquet" and a "Package AID:" field with the value "1234567890" (the first five digits are green). Below this is a "Hex:" label followed by "12 34 56 78 90". The second section, labeled "Applet AID:", contains a text field with the value "123456789001" (the first six digits are green) and a "Hex:" label followed by "12 34 56 78 90 01". At the bottom of the dialog are four buttons: a help button with a question mark, "< Back", "Next >", and "Finish", and a "Cancel" button.

Comme expliqué précédemment, il y a pour chaque AID au moins cinq octets obligatoires (ici en vert). On peut s'en contenter pour l'AID du package et rajouter un octet supplémentaire pour différencier les applets.

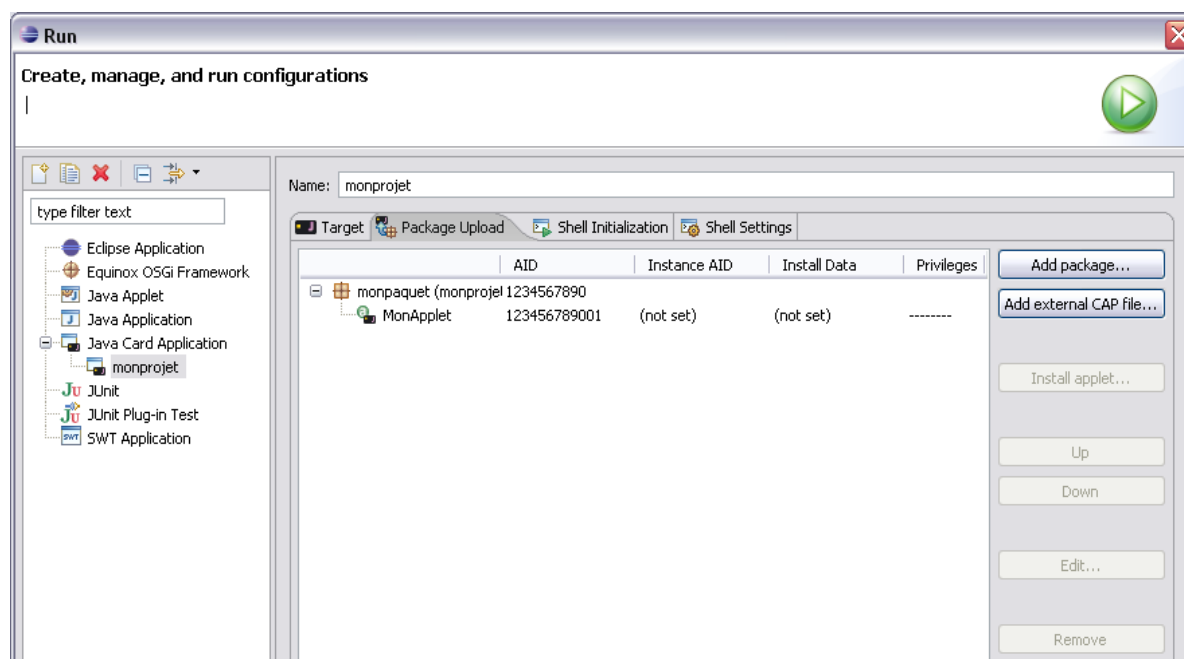
A présent on peut développer l'applet que l'on souhaite. Pour cela, la perspective « JCOP Development » est idéale. Nous verrons ultérieurement un exemple de code d'un applet. Une fois fait, voici la démarche à suivre pour compiler, charger dans la carte et exécuter le programme :

Allez dans le menu « Run » et créez une configuration « Java Card Application »

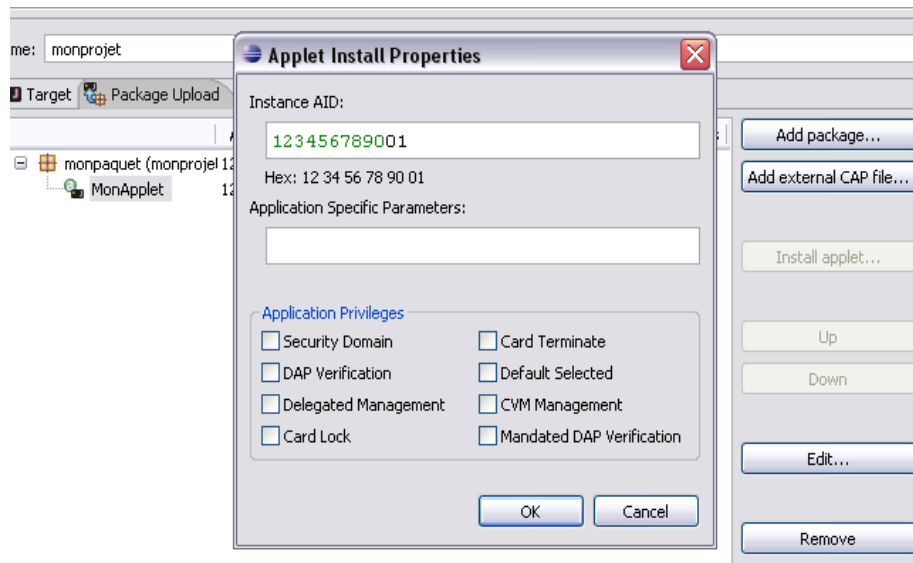


Le plug-in JCOP permet de simuler les applets directement sur l'ordinateur sans avoir besoin de charger votre programme sur une carte : « Java Card Simulation ». Si vous voulez au contraire charger votre applet sur une Java Card, sélectionnez le choix « Card Reader » en précisant le lecteur de carte utilisé.

Ensuite vérifiez que les AID sont tous bien définis dans l'onglet « Package Upload » :

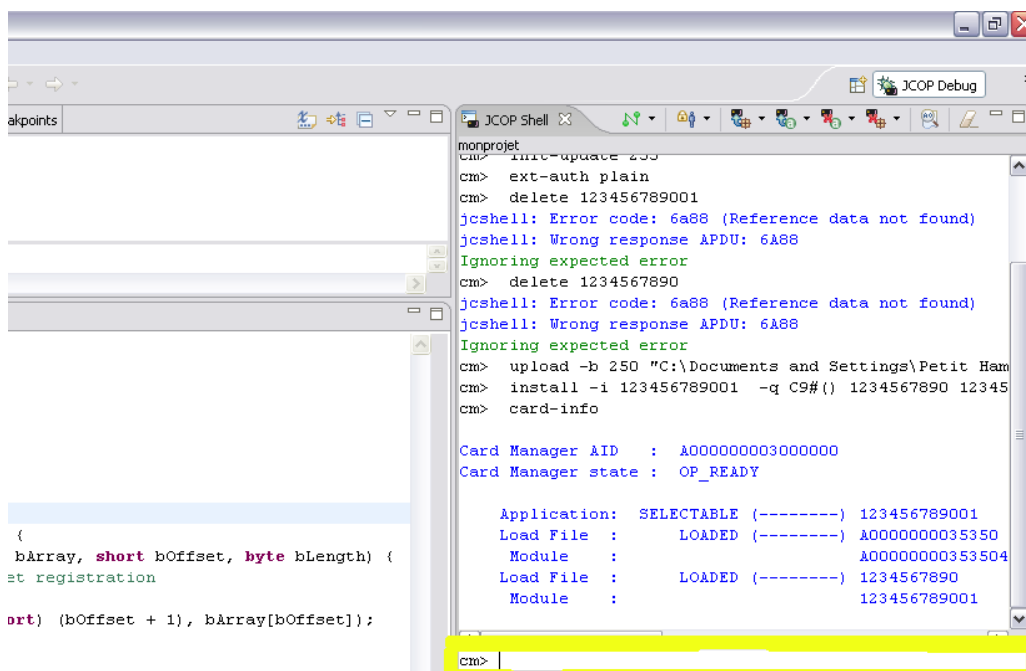


Il faut compléter le champ « Instance AID » en recopiant l'AID de l'applet. Cela permet à la machine virtuelle Java d'effectuer les liens nécessaires au bon fonctionnement du programme. Sélectionnez l'applet et cliquez sur « Edit ». Recopiez alors l'AID de l'applet et validez :



## Le Java Card Shell

A présent nous pouvons exécuter notre programme. Pour cela, la perspective « JCOP Debug » est conseillée. En effet cette perspective contient un shell de dialogue entre l'utilisateur et la carte (simulée ou réelle); c'est le « JCShell » (Java Card Shell).



Grâce à l'invite de commande du JCSHELL (surligné en jaune), on peut dialoguer avec la Java Card, et donc envoyer des commandes APDU.

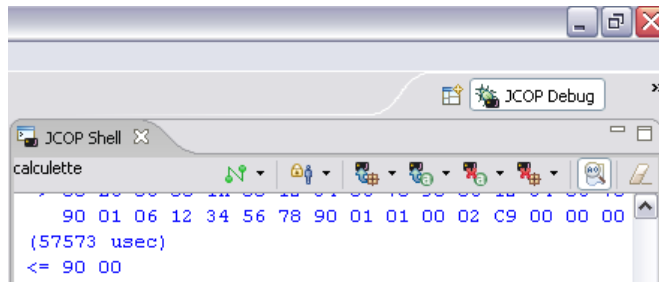
Pour rendre les requêtes plus compréhensibles, ce shell permet de ne pas être obligé de taper des commandes entièrement en hexadécimal. Selon ce que l'on souhaite faire, il existe une série d'alias correspondant à une commande précise, qui créera l'équivalent en hexadécimal. Voici une liste non exhaustive des commandes possibles du JCSHELL :

Commandes possibles à n'importe quel moment	
/cap-info	Affiche les informations sur un fichier CAP
/close	Ferme la connexion avec le terminal
/list-vars	Affiche la liste des variables interne du JCSHELL
/set-var	Crée un variable interne dans le JCSHELL
/mode	Change le mode d'opération (mode Trace par exemple)
/select	Sélectionne une applet avec son AID en paramètre
/send	Envoie une commande APDU en paramètre
/terminal	Se connecte au terminal donné ou donne les infos sur celui-ci si déjà connecté
quit	Ferme le JCSHELL
Commandes possibles uniquement si on est connecté à une carte : prompt : « cm> »	
upload	Charge un package au travers du fichier CAP dans la Java Card
install	Installe un applet et l'enregistre sous son AID
delete	Supprime un applet ou un package grâce à l'AID passé en paramètre
card-info	Affiche les informations sur les packages et les applets présents sur la carte

Pour de plus de détails sur les commandes possibles, vous pouvez consulter cette page web : [http://www.zurich.ibm.com/csc/infosec/jcop\\_tools/shellTutorial.html](http://www.zurich.ibm.com/csc/infosec/jcop_tools/shellTutorial.html)

Grâce au plug-in JCOP et à son Shell, il existe une interface graphique permettant d'effectuer quelques unes de ces opérations plus facilement.





En haut du « JCOP Shell » il y a une barre d'outils contenant des boutons réalisant les commandes suivantes (de gauche à droite) :



- Connexion à un terminal : le simulateur ou le lecteur de carte (« N » en vert)
- Reset de la carte
- Upload d'un package
- Install d'un applet
- Suppression d'un applet
- Suppression d'un package
- Mode « Trace APDUs». Affiche le détail des trames APDU échangées.

ATTENTION ! Il faut activer cette option si vous voulez tester votre applet et voir les APDUs. Par défaut cette option n'est pas activée et on ne peut voir que les codes d'erreur des réponses APDU si il y en a.

- Efface le shell

### III – Exemple d'application : la calculatrice

#### Description du code

Cet exemple permet de comprendre aisément le fonctionnement du code d'une applet Java Card. L'application présentée sert de simple calculatrice en remplissant les quatre fonctions d'addition, de soustraction, de multiplication et de division.

On rajoutera l'envoi d'une chaîne d'octets qui, en codage ASCII, représente le nom de l'application : « calculatrice ». Enfin on utilisera le champ de données d'une commande APDU en effectuant sur celle-ci une somme que l'on renverra.

```
package sample;

import javacard.framework.APDU;
import javacard.framework.Applet;
import javacard.framework.ISO7816;
import javacard.framework.ISOException;
import javacard.framework.Util;
```

Tout d'abord il faut importer toutes les classes nécessaires qui définissent l'applet Java Card pour qu'il fonctionne selon la norme ISO 7816. Ce sont les classes APDU, Applet et ISO7816. Enfin pour gérer les exceptions on importe la classe ISOException. Enfin la classe Util comporte des méthodes permettant d'effectuer de diverses opérations utiles.

```
public class Calculator extends Applet {
    private static final byte[] CALC = { //
        (byte) 0x63, // c
        (byte) 0x61, // a
        (byte) 0x6c, // l
        (byte) 0x63, // c
        (byte) 0x75, // u
        (byte) 0x6c, // l
        (byte) 0x61, // a
        (byte) 0x74, // t
        (byte) 0x72, // r
        (byte) 0x69, // i
        (byte) 0x63, // c
        (byte) 0x65  // e
    }; // Tableau de bytes (code ASCII) qui va être renvoyé si on envoie
    l'INS 0x00
```

L'applet Calculator a pour attribut un tableau d'octets qui sera renvoyé à l'application cliente quand le champ INS de la commande APDU aura pour valeur l'octet 00.

```
//methode d'installation de l'applet dans la carte
public static void install(byte[] bArray, short bOffset, byte bLength) {
    //Enregistrement de l'applet (methode register()) auprès du JCRE
    //Lors de la création de l'applet (new)
    new Calculator().register(bArray, (short) (bOffset + 1), bArray[bOffset]);
}
```

Méthode d'installation de l'applet dans la carte. Cette méthode crée donc un applet Calculator « new Calculator » et l'enregistre auprès du JCRE avec la méthode « register ». Cette méthode prend en paramètre une série d'octets qui sont les paramètres de l'applet. Ces paramètres sont ensuite utilisées pour l'enregistrement auprès du JCRE.

```
public void process(APDU apdu) {

    //pour verifier que cette applet n'est pas entrain d'être sélectionné
    if (selectingApplet()) {
        return;
    }

    //réception de la commande APDU
    byte[] buf = apdu.getBuffer();
    switch (buf[ISO7816.OFFSET_INS]) {
```

La méthode process gère les commandes APDU reçues depuis l'application cliente. Avant toute chose, elle vérifie si la commande en question n'est pas une sélection de l'applet ; sinon elle continue son traitement

L'APDU est stockée dans un tableau d'octets « buf » à partir de la commande « apdu.getBuffer() ». A partir de là, on peut sélectionner les champs souhaités à partir de la classe ISO7816. Ainsi on peut récupérer entre autres le champ INS, grâce au paramètre OFFSET\_INS.

Suivant la valeur récupérée, l'applet va effectuée une opération et renvoyer le résultat avec un code d'état :

```
case (byte) 0x00 :
    Util.arrayCopy(CALC, (short) 0, buf, (short) 0, (short) CALC.length);
    apdu.setOutgoingAndSend((short) 0, (short) CALC.length);
    return;
    // Ici, on copie le tableau (défini en haut) dans le buffer et on l'envoi sur
    la sortie
    // En précisant la taille du champ de données
```

INS = 00. Cette commande copie le tableau d'octets « CALC » dans le buffer d'APDU. On renvoie cette APDU avec la méthode « apdu.setOutgoingAndSend », en précisant la taille du champ de données « CALC.length ».

```

case (byte) 0x01 :
buf[0] = (byte) (buf[ISO7816.OFFSET_P1] + buf[ISO7816.OFFSET_P2]);
// Ici, on additionne P1 et P2
break;

```

INS = 01. Ici on demande d'effectuer une addition entre les deux paramètres P1 et P2. On place le résultat dans le premier octet de la réponse APDU : « buf[0] »

```

case (byte) 0x02 :
//Soustraction de P1 et P2
buf[0] = (byte) (buf[ISO7816.OFFSET_P1] - buf[ISO7816.OFFSET_P2]);
break;

```

INS = 02. Ici on demande d'effectuer une soustraction entre les deux paramètres P1 et P2.

```

case (byte) 0x03 :
//Multiplication
buf[0] = (byte) (buf[ISO7816.OFFSET_P1] * buf[ISO7816.OFFSET_P2]);
break;

```

INS = 03. Ici on demande d'effectuer une multiplication entre les deux paramètres P1 et P2.

```

case (byte) 0x04 :
//Division
if (buf[ISO7816.OFFSET_P2] == 0) { //Test de la division par 0
    ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);
    //Envoie de l'exception "mauvais paramètres"
}
buf[0] = (byte) (buf[ISO7816.OFFSET_P1] / buf[ISO7816.OFFSET_P2]);
break;

```

INS = 04. Ici on demande d'effectuer une division entre les deux paramètres P1 et P2. On teste également si P2 n'est pas égal à 0. Si c'est le cas on renvoie une exception du type « mauvais paramètre »

```

case (byte) 0x05 :
//Somme du champ de données de la commande APDU
short n = (short) (ISO7816.OFFSET_CDATA + apdu.setIncomingAndReceive());
byte sum = 0;
while (n-- > ISO7816.OFFSET_CDATA) {
    sum += buf[n];
}
buf[0] = sum;
break;

```

INS = 05. Ici on position le pointeur du buffer sur le champ de données de l'APDU reçue. On fait une boucle sur tout le champ de données pour en effectuer la somme octet par octet.

```

default :
//Envoie de l'exception "mauvais champ INS"
ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }
    //Pour tous les cas la réponse APDU a des données de 1 octet
    //Sauf pour le cas INS = 00
    apdu.setOutgoingAndSend((short) 0, (short) 1);
}

```

Si le champ INS prend une autre valeur que celle autorisée plus haut, on renvoie une exception de type « valeur INS non supportée ».

En dehors du cas INS=00, on renvoie à chaque fois une APDU avec un champ de données d'un octet : « apdu.setOutgoingAndSend((short) 0, (short) 1); »

## Tests avec le JCShell

Après une phase d'initialisation et de reconnaissance avec le simulateur JCOP, on charge le fichier .CAP qui contient notre package avec la commande « upload » :

```
cm> upload -b 250 "C:\Documents and Settings\robinc\Bureau\Hamster Work\Java
Card\eclipse\workspace\calcullette\bin\sample\javacard\sample.cap"
=> 80 E6 02 00 12 05 12 34 56 78 90 08 A0 00 00 00 .....4Vx.....
    03 00 00 00 00 00 00 00 .....
(1258 usec)
<= 00 90 00 ...
Status: No Error
=> 80 E8 00 00 FA C4 82 01 97 01 00 16 DE CA FF ED .....
    ...
    ... // Upload du package
    ...
    09 00 13 00 00 00 0F 05 06 04 0A 07 07 1A 06 04 .....
    06 04 33 12 25 06 00 ..3.%.
(7231 usec)
<= 00 90 00 ...
Status: No Error
Load report:
  411 bytes loaded in 0.0 seconds
  effective code size on card:
    + package AID      5
    + applet AIDs     13
    + classes          17
    + methods          203
    + statics          21
    + exports          0
-----
    overall           259 bytes
```

On installe ensuite notre applet avec la commande « install ». Cette opération enregistre également l'applet auprès du JCRC en communiquant les AID de l'applet et du package :

```
cm> install -i 123456789001 -q C9#() 1234567890 123456789001
=> 80 E6 0C 00 1A 05 12 34 56 78 90 06 12 34 56 78 .....4Vx...4Vx
    90 01 06 12 34 56 78 90 01 01 00 02 C9 00 00 00 ....4Vx.....
(3253 usec)
<= 90 00 ..
Status: No Error
cm> card-info
=> 80 F2 80 00 02 4F 00 00 .....O..
(563200 nsec)
<= 08 A0 00 00 00 03 00 00 00 01 9E 90 00 .....
Status: No Error
=> 80 F2 40 00 02 4F 00 00 ..@...O..
(557613 nsec)
<= 06 12 34 56 78 90 01 07 00 90 00 ..4Vx.....
Status: No Error
=> 80 F2 10 00 02 4F 00 00 .....O..
(917714 nsec)
<= 07 A0 00 00 00 03 53 50 01 00 01 08 A0 00 00 00 .....SP.....
```

```

    03 53 50 41 05 12 34 56 78 90 01 00 01 06 12 34    .SPA..4Vx.....4
    56 78 90 01 90 00                                Vx....
Status: No Error

Card Manager AID   : A000000003000000
Card Manager state : OP_READY

Application: SELECTABLE (-----) 123456789001
Load File   : LOADED (-----) A0000000035350 (Security Domain)
Module      : A000000003535041
Load File   : LOADED (-----) 1234567890
Module      : 123456789001

```

Ces deux premières opérations sont effectuées automatiquement par le plug-in JCOP. A l'exécution de l'applet nous nous retrouvons donc à ce stade.

La première chose à faire est de sélectionner l'applet sur lequel nous voulons travailler.

Pour cela il faut utiliser la commande « /select » suivit de son AID :

```

cm> /select 123456789001
=> 00 A4 04 00 06 12 34 56 78 90 01 00    .....4Vx...
(1017 usec)
<= 90 00    ..
Status: No Error

```

Comme expliqué précédemment, chaque réponse APDU se termine par deux octets qui désignent le code d'exécution de la commande. Une exécution sans erreur se traduit par un code de retour égal à 90 00 (cf. Annexe).

Commençons par demander de renvoyer le paramètre CALC de l'applet. Cette opération correspond à un champ INS = 00. On envoie donc l'APDU 00.00.00.00.00. Les valeurs de P1 et P2 n'ont pas d'importance ici :

```

cm> /send 00000000
=> 00 00 00 00    ....
(650083 nsec)
<= 63 61 6C 63 75 6C 61 74 72 69 63 65 90 00    calculatrice..
Status: No Error

```

L'APDU de retour nous renvoie bien le tableau d'octets demandé, suivi du code de retour 90 00 signifiant une bonne exécution de la commande. Sur la droite de l'écran se trouve une traduction de chaque octet suivant le code ASCII. On peut voir ici le mot demandé « calculatrice ».

Concernant l'addition, il faut donner la valeur 01 au champ INS. Testons l'addition 6+5. L'APDU envoyée sera donc 00.01.06.05 :

```
cm> /send 00010605
=> 00 01 06 05          ....
   (634438 nsec)
<= 0B 90 00            ...
Status: No Error
```

Le résultat renvoyé est bien correct : 0B (11 en hexadécimal).

Pour la soustraction INS doit valoir 02. Testons 10-8. L'APDU envoyée sera donc 00.02.0A.08 :

```
cm> /send 00020a08
=> 00 02 0A 08          ....
   (614883 nsec)
<= 02 90 00            ...
Status: No Error
```

Le résultat renvoyé est bien correct : 02

Pour la multiplication INS doit valoir 03. Testons 3x8. L'APDU envoyée sera donc 00.03.03.08 :

```
cm> /send 00030308
=> 00 03 03 08          ....
   (517943 nsec)
<= 18 90 00            ...
Status: No Error
```

Le résultat renvoyé est bien correct : 18 (24 en hexadécimal)

Pour la division INS doit valoir 04. Testons 40 / 8. L'APDU envoyée sera donc 00.04.28.08 :

```
cm> /send 00042808
=> 00 04 28 08          .. (.
   (662095 nsec)
<= 05 90 00            ...
Status: No Error
```

Le résultat renvoyé est bien correct : 05

Testons la division par zéro :

```
cm> /send 00042800
=> 00 04 28 00          .. (.
   (663492 nsec)
<= 6A 86                j.
Status: Incorrect parameters (P1,P2)
```

Le programme renvoie bien l'exception : Incorrect parameters



Testons la sommation du champ de données. Ici INS = 5. Il ne faut pas oublier que l'octet qui suit P2 signale la taille du champ de données. Nous allons envoyer les données 05.07.03. L'APDU envoyée sera donc 00.05.28.00.03.05.07.03 : (Ici les valeurs de P1 et P2 n'ont pas d'importance)

```
cm> /send 0005280003050703
=> 00 05 28 00 03 05 07 03      .. (.....
   (631644 nsec)
   <= 0F 90 00                  ...
Status: No Error
```

Le résultat renvoyé est bien correct : 0F (15 en hexadécimal) = 05+07+03

Testons une valeur de INS non autorisée : INS = 06

```
cm> /send 00062800
=> 00 06 28 00                  .. (
   (559847 nsec)
   <= 6D 00                      m.
Status: INS value not supported
```

L'exception est bien envoyée : valeur INS non supportée.

Testons un cas de débordement. Effectuons une multiplication dont le résultat est sur deux octets :  
 $40 \times 151 = 6040 = 1798$  en hexadécimal :

```
cm> /send 00032897
=> 00 03 28 97                  .. (
   (578845 nsec)
   <= 98 90 00                  ...
Status: No Error
```

Ici le JCRE ne détecte pas le problème, il se contente de prendre un seul octet, comme on le lui a demandé. Ici il renvoie l'octet de poids faible du résultat : 98

## CONCLUSION

Ce projet aura été une très bonne expérience. J'ai découvert un sujet original, une technologie très intéressante qui, bien que déjà très implantée dans le marché, a un grand avenir devant elle.

J'ai pu me rendre compte que cette technologie possède beaucoup d'attraits pour les personnes souhaitant développer des applications portables et sécurisées pour l'univers professionnel. Cette technologie a l'avantage d'être simple d'utilisation et possède une grande stabilité ainsi qu'une forte opacité du contenu face à l'environnement extérieur.

Tout au long de mes recherches sur cette technologie, j'ai pu non seulement comprendre le fonctionnement de ces applications embarquées, mais j'ai également pu enrichir mes connaissances sur Java en général. C'est en étudiant cette technologie que j'ai pu comprendre la véritable utilité de la machine virtuelle Java.

Durant ce projet j'ai dû faire face à diverses contraintes qui m'ont retardé dans la phase de développement d'applications. J'aurais beaucoup aimé avoir plus de temps pour comprendre et développer des applications plus complexes, qui traitent notamment de la sécurité, un aspect important de cette technologie. Je garde néanmoins une très bonne expérience de ce projet, qui m'apportera certainement un plus dans le monde professionnel. Les Java Card seront de plus en plus développées et améliorées, et avoir des connaissances sur le sujet sera sûrement un atout si l'occasion se présente.

# BIBLIOGRAPHIE

## Ouvrages consultés :

- « Tutorial - Installation, Activation and first steps with IBM JCOP Plug-in for Eclipse SDK » de Mobile Technologies

## Sites internet consultés :

- <http://java.sun.com/products/javacard/>
- <http://www.javacardforum.org/>
- [http://www.cru.fr/activites/cartes\\_puce/index](http://www.cru.fr/activites/cartes_puce/index)
- [http://www.zurich.ibm.com/csc/infosec/jcop\\_tools/shellTutorial.html](http://www.zurich.ibm.com/csc/infosec/jcop_tools/shellTutorial.html)
- <http://developers.sun.com/mobility/javacard/articles/javacard1/>
- <http://www-sop.inria.fr/oasis/personnel/Carine.Courbis/pubs/DEA/report/>
- [http://www.gemalto.com/press/gemplus/2005/rd/27-06-2005-javaone\\_fr.htm](http://www.gemalto.com/press/gemplus/2005/rd/27-06-2005-javaone_fr.htm)

# Annexes

Annexe : Codes définis par la norme ISO 7816 pour les champs CLA, INS, SW1 et SW2 d'une APDU

Valeurs pour le champ CLA :

0X, 1X	Accès aux données – Opérations de sécurité
X8, X9, de B0 à CF	Format utilisé pour le développement d'applications standard
AX, de D0 à FE	Instructions spécifiques aux vendeurs pour des applications standard
FF de 20 à 7F	Réservés

Valeurs pour le champ INS :

0E	ERASE BINARY
20	VERIFY
70	MANAGE CHANNEL
82	EXTERNAL AUTHENTICATE
84	GET CHALLENGE
88	INTERNAL AUTHENTICATE
A4	SELECT FILE
B0	READ BINARY
B2	RED RECORD(S)
C0	GET RESPONSE
C2	ENVELOPE
CA	GET DATA
D0	WRITE BINARY
D2	WRITE RECORD
D6	UPDATE BINARY
DA	PUT DATA
DC	UPDATE DATA
E2	APPEND RECORD

Les valeurs de P1 et P2 dépendent de la valeur de INS. Les champs SW1 et SW2 sont important car ils renseignent l'application cliente sur l'état de la commande APDU. Voici les valeurs possibles pour ces champs :

61 00	Response bytes remaining	SW_BYTES_REMAINING_00
67 00	Wrong length	SW_WRONG_LENGTH
69 82	Security condition not satisfied	SW_SECURITY_STATUS_NOT_SATISFIED
69 83	File invalid	SW_FILE_INVALID
69 84	Data invalid	SW_DATA_INVALID
69 85	Conditions of use not satisfied	SW_CONDITIONS_NOT_SATISFIED
69 86	Command not allowed	SW_COMMAND_NOT_ALLOWED
69 99	Applet selection failed	SW_APPLET_SELECT_FAILED
6A 80	Wrong data	SW_WRONG_DATA
6A 81	Function not supported	SW_FUNC_NOT_SUPPORTED
6A 82	File not found	SW_FILE_NOT_FOUND
6A 83	Record not found	SW_RECORD_NOT_FOUND
6A 84	Not enough memory in the file	SW_FILE_FULL
6A 86	Incorrect parameters (P1,P2)	SW_INCORRECT_P1P2
6B 00	Incorrect parameters (P1,P2)	SW_WRONG_P1P2
6C 00	Correct Expected Length (Le)	SW_CORRECT_LENGTH_00
6D 00	INS value not supported	SW_INS_NOT_SUPPORTED
6E 00	CLA value not supported	SW_CLA_NOT_SUPPORTED
6F 00	No precise diagnosis	SW_UNKNOWN

Valeurs spécifiques de SW que pourrait retourner le Card Manager en cas de problème avec la JCVM :

62 83	Selected applet blocked
63 00	Cryptogram not verified
63 10	More data available
65 81	Memory failure
67 00	Incorrect length
6A 88	No element matches the search criteria
6C XX	Only XX bytes of data are available
6F 80	Stack overflow
90 00	Command process without error
92 40	Memory problem
94 02	Access out of range
94 84	Algorithm not supported
94 85	Invalid key check value
9F XX	XX bytes of data available