ED 1 : Interaction avec la carte à puce selon les protocoles ISO 7816-3 et ISO 7816-4

Exemple de la carte vitale ou tout autre carte de votre porte-feuille

Samia Bouzefrane & Julien Cordry

Partie I : Introduction

1. Introduction

L'ISO 7816 « Idenfication cards – Integrated circuit cards with contacts » a été publié par l'organisation internationale de standardisation (ISO, International Organisation for Standardisation). C'est le plus important standard définissant les caractéristiques des cartes à puce qui fonctionnent avec un contact électrique. Sachant que 15 normes sont proposées pour les cartes à contact, nous décrivons brièvement ici uniquement les 7 premières normes.

1.1 ISO 7816-1

Cette norme définit les caractéristiques physiques des cartes à puce à contact : la géométrie, la résistance, les contacts, etc.

1.2 ISO 7816-2

Cette norme spécifie le dimensionnement physique (extérieur) des contacts de la puce. Deux des huit contacts réservés à une utilisation future (RFU) sont redéfinis pour l'utilisation USB dans la norme ISO 7816-12.

1.3 ISO 7816-3

Cette norme définit l'interface électrique et les protocoles de transmission :

- les protocoles de transmission (TPDU, Transmission Protocol Data Unit) : T=0 : protocole orienté octet, T1 : protocole orienté paquet, T=14 : réservé pour les protocoles propriétaires,
- la sélection d'un type de protocole,
- la réponse à un reset (ATR, ou Answer To Reset en anglais) qui correspond aux données envoyées par la carte immédiatement après la mise sous tension,
- les signaux électriques, tels que le voltage, la fréquence d'horloge et la vitesse de communication.

1.4 ISO 7816-4

Cette norme vise à assurer l'interopérabilité des échanges. Elle définit les messages APDU (Application Protocol Data Units), par lesquels les cartes à puce communiquent avec le lecteur. Les échanges s'effectuent en mode client-serveur, le terminal ayant toujours l'initiative de communication.

1.5 ISO 7816-5

Cette norme définit le système de numérotation et les procédures d'enregistrement et d'attribution des identifiants des applications (AID, ou *Application IDentifier*). Un unique AID est associé à chaque application et à certains fichiers sur la carte. Ils sont représentés par des tableaux d'octets de taille allant de cinq à seize. Les cinq premiers octets représentent le numéro d'enregistrement du fournisseur d'application (RID, *Registered Application Provider IDentifier* en anglais) qui est attribué par la Copenhagen Telephone Company Ltd. Ils sont suivis par l'identifiant optionnel PIX (*Proprietary Application Identifier eXtension*) d'une longueur allant jusqu'à 11 octets.
L'identifiant RID est le même pour le paquetage et l'applet, mais le PIX doit être différent.

1.6 ISO 7816-6

Cette norme spécifie des éléments de données inter-industrie pour les échanges, tels que le numéro du porteur de carte, sa photo, sa langue, la date d'expiration, etc.

1.7 ISO 7816-7

Cette norme définit les commandes inter-industrie pour langage d'interrogation de carte structurée (SCQL).

2. ATR (Answer To Reset) défini dans l'ISO 7816-3

Dès que la carte est mise sous tension, elle envoie un message de réponse d'initialisation appelé ATR, il peut atteindre une taille maximale de 33 octets. Il indique à l'application cliente les paramètres nécessaires pour établir une communication avec elle. Il fournit un nombre varié de paramètres liés à la carte et au protocole de transmission utilisé :

– Le protocole de transport ;
– Taux de transmission des données ;
– Numéro de série de la puce …

Le premier octet noté TS ="3F" pour convention indirecte ou "3B" pour convention directe.

3. Echange de commandes avec le lecteur de carte à puce tel que défini dans l' ISO 7816-4

La communication entre l'hôte et la carte est half-duplex. Elle se fait à l'aide de paquets appelés APDU (Application Protocol Data Units) en respectant le protocole de l'ISO 7816-4. Un APDU contient une commande ou une réponse. Le mode Maître/Esclave est utilisé. Ainsi la carte joue un rôle passif et attend une commande APDU à partir de l'hôte. Elle exécute l'instruction spécifiée dans la commande et retourne une réponse APDU.

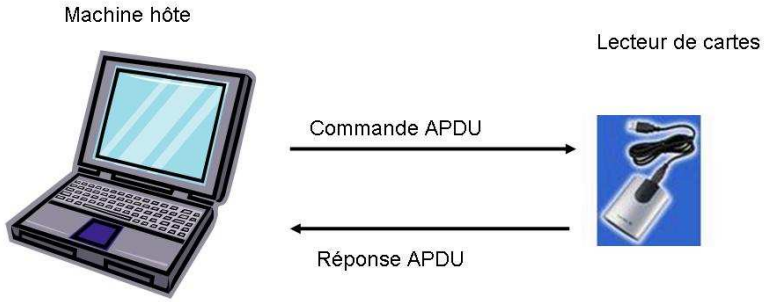

Figure 1 : Le modèle de communication de la carte à puce

3.1. Format des commandes APDU

Commande APDU						
Entête obligatoire				**Corps optionnel**		
CLA	**INS**	**P1**	**P2**	**Lc**	**Data field**	**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 (1 octet) et P2 (1 octet): paramètres de l'instruction
- Lc (1 octet): nombre d'octets présents dans le champ données de la commande
- Data field (octets dont le nombre est égal à la valeur de Lc): une séquence d'octets dans le champ données de la commande

3.2. Format des réponses APDU

Réponse APDU		
Corps optionnel	**Partie obligatoire**	
Data field	**SW1**	**SW2**

- Data field (longueur variable): une séquence d'octets reçus dans le champ données de la réponse
- SW1 (1 octet) et SW2 (1 octet): Status words (Mots d'état)—état de traitement par la carte

4. Exemples de cartes

Le tableau suivant donne des exemples de commandes APDU utilisées dans le monde de la carte.

Champ de la commande APDU	**Valeurs**
CLA	BC = cartes de crédit françaises, cartes vitales françaises, A0 = cartes SIM (téléphonie) 00 = cartes Monéo (porte-monnaie en France), Mastercard, Visa
INS	20 = vérification du PIN, B0 = Lecture B2 = Lecture de record D0 = Ecriture DC = Ecriture de record A4 = Sélection du répertoire (directory) C0 = Demander une réponse (get an anwer)
P1, P2	paramètres contenant des adresses à lire
LEN	longueur prévue pour la réponse ou bien longueur de l'argument de l'instruction
ARG	contient LEN octets (octets à écrire, PIN à vérifier, etc.)

La réponse APDU sert à accuser réception la commande APDU envoyée par le terminal. Ainsi, la carte répond en envoyant le code instruction INS, suivi de données de longueur LEN en terminant par SW1 et SW2 (0x90 0x00 lorsque la commande s'est déroulée avec succès). En cas d'échec, seuls les champs SW1 et SW2 seront envoyés au terminal avec les codes d'erreur suivants :

0x6E 0x00 CLA error
0x6D 0x00 INS error
0x6B 0x00 P1, P2 error
0x67 0x00 LEN error
0x98 0x04 Bad PIN
0x98 0x08 Unauthorized Access
0x98 0x40 Card blocked
...

Partie II : Interaction avec la carte à puce
Les premières commandes en utilisant un script Perl

Sous l'environnement **Linux** :

Etape 1 :
- télécharger **pcsc-perl-1.4.4.tar.gz** à partir de l'adresse suivante :
`http://ludovic.rousseau.free.fr/softwares/pcsc-perl`
pcsc-perl est une interface Perl permettant l'accès à la librairie PC/SC smartcard.

- décompresser le fichier:
```
$tar xvfz pcsc-perl-1.4.4.tar.gz
$ls
pcsc-perl-1.4.4
$
```
- et l'installer comme suit :
```
$cd pcsc-perl-1.4.4
pcsc-perl-1.4.4>$perl Makefile.PL
pcsc-perl-1.4.4>$make
//insérer la carte
pcsc-perl-1.4.4>$make test
pcsc-perl-1.4.4>$sudo make install
```
En cas d'erreur à l'installation, assurez-vous que la librairie pcsc-lite figure bien dans les librairies système.

Etape 2 :
- télécharger **pcsc-tools-1.4.7.tar.gz** à partir de l'adresse suivante :
`http://ludovic.rousseau.free.fr/softwares/pcsc-tools`

- décompresser le fichier :
```
$tar xvfz pcsc-tools-1.4.7.tar.gz
$ls
pcsc-tools-1.4.7
$
```

pcsc-tools contient des outils qui peuvent être utiles à un utilisateur de PC/SC.

Ces outils fournissent :

- **pcsc_scan** : qui scanne régulièrement chaque lecteur PC/SC relié à l'hôte. Si une carte est insérée ou enlevée une ligne est affichée en précisant la date, le type de lecteur, le type de carte et l'état de la carte (retirée ou insérée).

- **ATR_analysis** : script Perl (appelé par défaut par pcsc-scan) est utilisé pour analyser l'ATR de la carte à puce.

- **smartcard_list.txt** : contient la liste de ATR pour certaines cartes. Cette liste est utilisée par ATR_analysis pour trouver un modèle de carte correspondent à l'ATR.

- **scriptor** : script Perl servant à envoyer des commandes à la carte. On peut utiliser un fichier de commandes.

- **gscriptor** : script Perl servant à envoyer des commandes en utilisant une interface graphique.

Exercice :

Accédez au répertoire `pcsc-tools-1.4.7`. Créer un fichier de commandes `carte.txt` qui contient :

```
reset
```

Insérer votre carte vitale dans le lecteur de carte et lancez `scriptor carte.txt`. L'ATR est renvoyée. Quelle est la taille de l'ATR ?

Complétez votre fichier de commandes au fur et à mesure à l'aide de commandes APDU afin de :

- lire la dernière adresse accessible, soit X cette adresse
- soustraire 1F00 à l'adresse X pour lire 256 octets à partir de la nouvelle adresse obtenue Y. Trouvez une séquence d'octets qui commencent par la séquence "3F FF F" pour repérer votre année de naissance. Les 2 octets qui suivent contiennent dans l'ordre le mois et le jour de naissance en faisant abstraction des deux premiers bits les plus significatifs du premier octet. La date de naissance étant codée en DCB (Décimal Codé Binaire).
- soustraire 0200 à l'adresse X et lire 256 octets à partir de la nouvelle adresse.

Partie III : Interaction avec la carte à l'aide d'un programme Java 1.6

1. JSR 268

JSR 268 définit une API Java pour permettre la communication avec les cartes à puce en utilisant le protocole ISO 7816-4. Il permet ainsi aux applications écrites en Java 1.6 d'interagir avec des applications qui tournent sur la carte à puce, en utilisant Linux ou Windows. Attention utilisez la version JDK 1.6.beta2 qui implante complètement JSR 268.

Description du Package `javax.smartcardio`

Classe et Constructeur	Méthodes de la classe

ATR : Réponse à un Reset Answer To Reset
 	
`ATR``(byte[] atr)`
construit un ATR à partir d'un tableau d'octets.
	`boolean` **`equals`**`(Object obj)`
	Compares the specified object with this ATR for equality.
	
	`byte[]`**`getBytes`**`()`
	Returns a copy of the bytes in this ATR.
	
	`String` **`toString`**`()`
	Returns a string representation of this ATR.
	…

Card : Etablissement de connexions
 	
Protected **Card**()
construit un objet Card.
	`abstract void` **`disconnect`**`(boolean reset)`
	déconnexion de la carte.
	
	`abstract ATR` **`getATR`**`()`
	retourne l'ATR de la carte.
	
	`abstract String` **`getProtocol`**`()`
	retourne le protocole de cette carte.
	
	`abstract CardChannel` **`getBasicChannel`**`()`
	retourne le CardChannel d'un canal logique de base
	…

CardChannel : Connexion logique à la carte à puce
 	
`protected` **`CardChannel`**`()`
construit un objet CardChannel
	`abstract void` **`close`**`()`
	ferme le CardChannel
	
	`abstract ResponseAPDU` **`transmit`**`(CommandAPDU command)`
	envoie la commande APDU vers la carte et retourne une réponse APDU.
	
	…

CardTerminal :**Création d'un lecteur de carte**
 	
`protected` **`CardTerminal`**`(`TerminalFactory factory`)`
construit un nouvel objet CardTerminal
	`abstract Card` **`connect`**`(String protocol)`
	établit une connexion avec la carte.
	
	`abstract String` **`getName`**`()`
	retourne le nom du lecteur.
	
	`abstract boolean` **`isCardPresent`** `()`
	retourne vrai si la carte est présente dans le lecteur, faux sinon.
	
	…

CommandAPDU : Une commande APDU tel que	`byte[]`**`getBytes`**`()`

défini dans ISO 7816-4
 	
`CommandAPDU`(byte[] apdu)
construit une commande APDU à partir d'un tableau d'octets.
 	
`CommandAPDU`(byte cla, byte ins, byte p1, byte p2)
construit une commande APDU à partir des 4 octets de l'entête.
 	
etc.
	retourne une copie des octets de l'APDU.
	
	`byte` **`getCLA`**`()`
	retourne la valeur de CLA.
	
	`byte` **`getINS`**`()`
	retourne la valeur en octet de INS.
	
	`byte` **`getP1`**`()`
	retourne la valeur en octet du paramètre P1.
	
	`byte` **`getP2`**`()`
	retourne la valeur en octet du paramètre P2.
	
	`int getNe``()`
	retourne le nombre maximal d'octets attendus en réponse.
	
	etc.

ResponseAPDU : la réponse APDU tel que défini dans ISO 7816-4 **`ResponseAPDU`**(byte[] apdu) construit une réponse APDU à partir d'un vecteur d'octets contenant l'APDU complète (données et SW1 et SW2).	**`byte [] getBytes`**`()` retourne une copie de l'APDU en octets. **`byte [] getData`**`()` retourne une copie en octets des données envoyées en réponse à la commande APDU. **`byte getSW1`**`()` retourne la valeur en octet de l'état SW1. **`byte getSW2`**`()` retourne la valeur en octet de l'état SW2. etc.

TerminalFactory : une fabrique pour les objets CardTerminal	`CardTerminal`
	`getTerminal``(String name)`
	retourne le terminal avec le nom spécifié ou null si le terminal spécifié n'existe pas.
	
	`static TerminalFactory` **`getDefault`**`()`
	retourne une instance de TerminalFactory par défaut.
	etc.

2. Exercices

2.1 Exercice 1

Après installation de la JDK 1.6.beta2 sur votre machine, utilisez le package `javax.smartcardio` pour écrire un programme Java (`Vitale.java`) qui se contente d'afficher l'ATR et qui envoie une commande APDU vers votre carte vitale pour lire la dernière adresse accessible.

Pour connaître le nom du lecteur utilisé, tapez la commande suivante sous Linux :

```
$tail –f /var/log/messages
$
```

Sous Windows, voir au niveau des registres internes.

Un certain nombre de méthodes, proposées par Julien Cordry, sont disponibles dans les classes `Tools` et `Address` pour faciliter respectivement la manipulation d'APDUs et celle des adresses.

2.2 Exercice 2

Compléter le programme précédent afin de :

- lire la dernière adresse accessible, soit X cette adresse
- soustraire 1F00 à l'adresse X pour lire 256 octets à partir de la nouvelle adresse obtenue Y. Trouvez une séquence d'octets qui commencent par la séquence "3F FF F" pour repérer votre

année de naissance. Les 2 octets qui suivent contiennent dans l'ordre le mois et le jour de naissance en faisant abstraction des deux premiers bits les plus significatifs du premier octet. La date de naissance étant codée en DCB (Décimal Codé Binaire).

- soustraire 0200 à l'adresse X et lire 256 octets à partir de la nouvelle adresse.

Précisez le contenu des APDUs impliqués dans ce traitement à chaque étape.

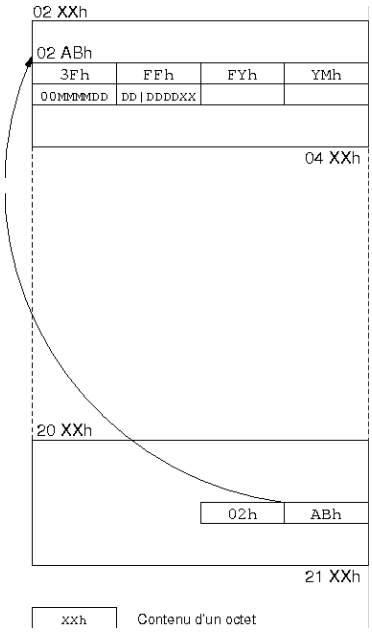

Figure 2 : La structure de la carte vitale

Partie IV : Solution

Exercice 1 :

```
package vitale;

import javax.smartcardio.*;
import vitale.Tools;

public class Vitale {

    /**
    * @param args */
    public static void main(String[] args) {
       TerminalFactory tf = TerminalFactory.getDefault();
       CardTerminal cad = tf.getTerminal("Towitoko Chipdrive Micro 00 00");
       try {
              Card c = cad.connect("*");
              ATR atr = c.getATR();
              System.out.println("ATR : " + atr.toString());
              CardChannel cc = c.getBasicChannel();

              // Pour obtenir la derniere adresse
              byte [] apdu = {
                     (byte) 0xBC, (byte) 0xC0, (byte) 0x00, (byte)0x00, (byte) 0x08
                     };
              CommandAPDU capdu = new CommandAPDU(apdu);
              System.out.println("==>   :   "   +   capdu.toString()   +   "\n==>   "   +
Tools.ba2s(capdu.getBytes()));
              ResponseAPDU r = cc.transmit(capdu);
              System.out.println("<==    :    "    +    r.toString()    +    "\n"    +
Tools.ba2s(r.getBytes()) + "\n");
              byte [] laddress = {
                     r.getBytes()[2], r.getBytes()[3]
              };

       } catch (Exception e){ e.printStackTrace();   }
   } // fin du main
} // fin de la classe
```

Exercice 2 :

```
package vitale;

import javax.smartcardio.*;
import vitale.Tools;

public class Vitale {

    /**
    * @param args */
    public static void main(String[] args) {
       TerminalFactory tf = TerminalFactory.getDefault();
       CardTerminal cad = tf.getTerminal("Towitoko Chipdrive Micro 00 00");
       try {
              Card c = cad.connect("*");
              ATR atr = c.getATR();
              System.out.println("ATR : " + atr.toString());
              CardChannel cc = c.getBasicChannel();

              // Pour obtenir la derniere adresse
              byte [] apdu = {
                     (byte) 0xBC, (byte) 0xC0, (byte) 0x00, (byte)0x00, (byte) 0x08
                     }; // on ne peut pas lire moins de 8 octets
              CommandAPDU capdu = new CommandAPDU(apdu);
              System.out.println("==>   :   "   +   capdu.toString()   +   "\n==>   "   +
Tools.ba2s(capdu.getBytes()));
              ResponseAPDU r = cc.transmit(capdu);
```

```
            System.out.println("<==    :    "    +    r.toString()    +    "\n"    +
Tools.ba2s(r.getBytes()) + "\n");
            byte [] laddress = {
                  r.getBytes()[2], r.getBytes()[3]
            };
            Address lastaddress = new Address(laddress);

// On recherche la premiere adresse : on soustrait 1F00 de la derniere adresse
            // NB : les addresses marchent par quartet et non par octet
            byte [] sub = {
                  (byte) 0x1F, (byte) 0x00
            };
            Address firstaddress = lastaddress.sub(new Address(sub));
            System.out.println("Premiere        adresse        :        "        +
Tools.ba2s(firstaddress.getAddress()) +
                        "\n256 premiers octets :");

            // Lecture des 256 premiers octets
            apdu[1] = (byte) 0xB0;
            apdu[2] = firstaddress.getAddress()[0];
            apdu[3] = firstaddress.getAddress()[1];
            apdu[4] = (byte) 0x00; // 256 en hexa = 100,
                                   // 100 sur 1 octet=00
            capdu = new CommandAPDU(apdu);
            System.out.println("==>   :   "   +   capdu.toString()   +   "\n==>   "   +
Tools.ba2s(capdu.getBytes()));
            r = cc.transmit(capdu);
            System.out.println("<==    :    "    +    r.toString()    +    "\n"    +
Tools.contentRAPDU(firstaddress, r.getBytes()) + "\n");
// Lecture des 256 derniers octets : on regarde 0200 avant la derniere adresse
            sub[0] = (byte) 0x02;
            Address lastwords = lastaddress.sub(new Address(sub));
            System.out.println("256   derniers   octets   a   partir   de   :   "   +
Tools.ba2s(lastwords.getAddress()));
            apdu[2] = lastwords.getAddress()[0];
            apdu[3] = lastwords.getAddress()[1];
            apdu[4] = (byte) 0x00;
            capdu = new CommandAPDU(apdu);
            System.out.println("==>   :   "   +   capdu.toString()   +   "\n==>   "   +
Tools.ba2s(capdu.getBytes()));
            r = cc.transmit(capdu);
            System.out.println("<==    :    "    +    r.toString()    +    "\n"    +
Tools.contentRAPDU(lastwords, r.getBytes()) + "\n");
      } catch (Exception e){
            e.printStackTrace();
      }
   }
}
```

Résultat d'exécution :

```
ATR : ATR: 9 bytes
==> : CommmandAPDU: 5 bytes, nc=
==> BC C0 00 00
08
<== : ResponseAPDU: 10 bytes, SW
00 00 21 88
00 00 0E 08
90 00

Premiere adresse : 02 88
256 premiers octets :
==> : CommmandAPDU: 5 bytes, nc=
==> BC B0 02 88
00
<== : ResponseAPDU: 258 bytes, S
02 88 : 2C 9A 05 85
02 90 : 26 59 85 A0
02 98 : 04 00 7F FF
02 A0 : 33 FF FF FF
02 A8 : 00 00 00 00
```

```
02 B0 : 03 02 02 7F
02 B8 : 09 94 06 64
02 C0 : 35 41 98 09
02 C8 : 04 DD 44 14
02 D0 : 06 04 11 01
02 D8 : 00 00 00 00
02 E0 : 00 00 03 FF
02 E8 : 22 FF FF FF
02 F0 : 3F FF F6 50
02 F8 : 04 5C 43 FF
03 00 : 3F FF FF FF
03 08 : 3F FF FF FF
03 10 : 3F FF FF F9
03 18 : 21 6A 42 00
03 20 : 00 00 00 00
03 28 : 00 06 C4 FA
03 30 : 3A 29 A4 17
03 38 : 05 00 00 00
03 40 : 00 00 00 00
03 48 : 00 00 01 FF
03 50 : 3F FF F9 70
03 58 : 20 4C 58 10
03 60 : 12 31 FF FF
03 68 : 3F FF FF FF
03 70 : 3F FF FF F6
03 78 : 21 99 5D 32
03 80 : 2E 00 00 00
03 88 : 00 06 C4 FA
03 90 : 3A 29 A4 17
03 98 : 05 00 00 00
03 A0 : 00 00 00 00
03 A8 : 00 00 01 FF
03 B0 : 20 00 00 00
03 B8 : 00 00 00 00
03 C0 : 13 80 00 00
03 C8 : 00 A2 3F FF
03 D0 : 20 00 00 00
03 D8 : 00 00 00 00
03 E0 : 13 80 00 00
03 E8 : 00 02 20 3F
03 F0 : 09 94 06 64
03 F8 : 35 41 98 09
04 00 : 04 DD 44 14
04 08 : 06 05 11 01
04 10 : 00 00 00 00
04 18 : 00 00 03 FF
04 20 : 22 FF FF FF
04 28 : 3F FF F0 10
04 30 : 10 98 58 10
04 38 : 12 31 FF FF
04 40 : 3F FF FF FF
04 48 : 3F FF FF FA
04 50 : 01 73 92 E0
04 58 : 20 00 00 00
04 60 : 00 06 C4 FA
04 68 : 3A 29 A4 17
04 70 : 05 00 00 00
04 78 : 00 00 00 00
04 80 : 00 00 01 FF

SW : 9000

256 derniers octets a partir de
==> : CommmandAPDU: 5 bytes, nc=
==> BC B0 1F 88
00
<== : ResponseAPDU: 258 bytes, S
1F 88 : FF FF FF FF
1F 90 : FF FF FF FF
1F 98 : FF FF FF FF
1F A0 : FF FF FF FF
1F A8 : FF FF FF FF
1F B0 : FF FF FF FF
1F B8 : FF FF FF FF
1F C0 : FF FF FF FF
1F C8 : FF FF FF FF
1F D0 : FF FF FF FF
```

```
1F D8 : FF FF FF FF
1F E0 : FF FF FF FF
1F E8 : FF FF FF FF
1F F0 : FF FF FF FF
1F F8 : FF FF FF FF
20 00 : FF FF FF FF
20 08 : FF FF FF FF
20 10 : FF FF FF FF
20 18 : FF FF FF FF
20 20 : FF FF FF FF
20 28 : FF FF FF FF
20 30 : FF FF FF FF
20 38 : FF FF FF FF
20 40 : FF FF FF FF
20 48 : FF FF FF FF
20 50 : FF FF FF FF
20 58 : FF FF FF FF
20 60 : FF FF FF FF
20 68 : FF FF FF FF
20 70 : FF FF FF FF
20 78 : FF FF FF FF
20 80 : FF FF FF FF
20 88 : FF FF FF FF
20 90 : FF FF FF FF
20 98 : FF FF FF FF
20 A0 : FF FF FF FF
20 A8 : FF FF FF FF
20 B0 : FF FF FF FF
20 B8 : FF FF FF FF
20 C0 : FF FF FF FF
20 C8 : FF FF FF FF
20 D0 : 3F FF C4 A8
20 D8 : 00 C0 84 88
20 E0 : 00 80 84 28
20 E8 : 20 3F C3 F0
20 F0 : 00 C0 43 D0
20 F8 : 20 C0 03 B0
21 00 : 00 80 43 50
21 08 : 20 80 02 F0
21 10 : 00 3F C2 B8
21 18 : 2D 3F C2 A0
21 20 : 2C FF C2 98
21 28 : FF FF FF FF
21 30 : 3F 21 28 E7
21 38 : 3F 02 98 FA
21 40 : 3F 02 88 F4
21 48 : 3F 02 60 ED
21 50 : 3F 02 48 F6
21 58 : 3F 02 30 FE
21 60 : 3F 86 EE D7
21 68 : 3F 02 10 3C
21 70 : 3F FF 00 6F
21 78 : 09 AA AC F9
21 80 : 5C 8E 9F FF

SW : 9000

Press any key to continue...
```

Partie V : Annexe

Classes à utiliser :

```
// >>> Classe Tools
package vitale;

public class Tools {

	private static String b2s (byte b) {
		StringBuffer sb = new StringBuffer();
		sb.append (quartet2c((byte)((b & 0xF0) >> 4)));
		sb.append (quartet2c((byte)(b & 0x0F)));
		return(sb.toString());
	}

	/**
	 * @param byteArray
	 * @return the String value for the byte array
	 */
	public static String ba2s (byte [] byteArray) {
		StringBuffer sb = new StringBuffer();
		for (int i = 0; i < byteArray.length; i++) {
			sb.append(b2s(byteArray[i]));
			if (i % 4 == 3) {
				sb.append('\n');
			} else {
				sb.append(' ');
			}
		}
		return (sb.toString());
	}

	/**
	 *
	 * @param add : the address for the first byte
	 * @param rapdu : the rAPDU that needs to be
	 * @return the String representation of the apdu indexed with the addresses
	 */
	public static String contentRAPDU (Address address, byte [] rapdu) {
		StringBuffer sb = new StringBuffer();
		Address local = address;
		byte [] badd = {(byte) 0x08};
		Address add = new Address(badd);
		int i;
		for (i = 0; i < rapdu.length - 5; i+=4) {
			sb.append (ba2s(local.getAddress()) + ": ");
			byte [] ba = {rapdu[i], rapdu[i+1], rapdu[i+2], rapdu[i+3]};
			sb.append(ba2s(ba));
			local = local.add(add);
		}
		if (i < rapdu.length - 2) {
			sb.append (ba2s(local.getAddress()) + ": ");
		}
		for (; i < rapdu.length - 2; i++) {
			byte [] ba = {rapdu[i]};
			sb.append(ba2s(ba));
		}
		sb.append("\nSW : " + b2s(rapdu[i]) + b2s(rapdu[i+1]));
		return sb.toString();
	}

	private static char quartet2c (byte b) {
		char c;
		switch (b) {
		case 0 : c = '0'; break;
		case 1 : c = '1'; break;
		case 2 : c = '2'; break;
```

```
                case 3 : c = '3'; break;
                case 4 : c = '4'; break;
                case 5 : c = '5'; break;
                case 6 : c = '6'; break;
                case 7 : c = '7'; break;
                case 8 : c = '8'; break;
                case 9 : c = '9'; break;
                case 10 : c = 'A'; break;
                case 11 : c = 'B'; break;
                case 12 : c = 'C'; break;
                case 13 : c = 'D'; break;
                case 14 : c = 'E'; break;
                case 15 : c = 'F'; break;
                default : c = '.'; break;
                }
                return (c);
        }
}

// >>> Classe Address
package vitale;

import java.util.Vector;

public class Address {
        public byte[] address;

        /**
         * @param address
         */
        public Address(byte[] address) {
                super();
                this.address = address;
        }

        /**
         * @return the address
         */
        public byte[] getAddress() {
                return address;
        }

        /**
         * @param address
         */
        public void setAddress(byte[] address) {
                this.address = address;
        }

        /**
         * @param address : the address that is subtracted from <i>this</i>
         * @throws Exception
         */
        public Address sub(Address address) throws Exception {
                Vector<Byte> v = new Vector<Byte>();
                byte r = (byte) 0;
                if (address.leq(this)) {
                        for (int i = this.address.length - 1; i >= 0; i--) {
                                int a = (this.address[i] & 0xFF);
                                int b;
                                if (i - (this.address.length - address.address.length) >=
0) {
                                        b = (address.address[i - (this.address.length -
address.address.length)]
                                                        & 0xFF);
                                } else {
                                        b = (byte) 0;
                                }
```

```
                        if (a >= (b + r)) {
                              v.insertElementAt(new Byte((byte) (a - (b + r))),
0);
                              r = (byte) 0;
                        } else {
                              v.insertElementAt(new Byte((byte) ((256 + a) - (b +
r))), 0);
                              r = (byte) 1;
                        }
                  }
                  int blanks = 0;
                  while ((v.size() != 0) && (v.elementAt(0).byteValue() == (byte)
0)) {
                        v.remove(0);
                        blanks++;
                  }
                  Byte[] Ba = new Byte[this.address.length - blanks];
                  v.toArray(Ba);
                  byte[] ba = new byte[Ba.length];
                  for (int i = 0; i < ba.length; i++) {
                        ba[i] = Ba[i].byteValue();
                  }
                  return (new Address(ba));
            } else {
                  throw (new Exception("subtraction error"));
            }
      }

      /**
       * @param address : the address that should be added to this
       * @return : this + address
       */
      public Address add (Address address) {
      Address ret;
      Vector <Byte> c = new Vector <Byte> ();
      int r = (int) 0;
            for (int i = this.address.length - 1; i >= 0; i--) {
                  int a = (this.address[i] & 0xFF);
                  int b;
                  if (i - (this.address.length - address.address.length) >= 0) {
                        b = (address.address[i - (this.address.length -
address.address.length)]
                                                & 0xFF);
                  } else {
                        b = (byte) 0;
                  }
                  if ((a + b + r) < 256) {
                        c.insertElementAt(new Byte ((byte)(a + b + r)), 0);
                        r = (byte) 0;
                  } else {
                        c.insertElementAt(new Byte ((byte)(a + b + r - 256)), 0);
                        r = (byte) 1;
                  }
            }
            if (r == 1) {
                  c.insertElementAt(new Byte ((byte) 1), 0);
            }
            Byte[] Ba = new Byte[c.size()];
            c.toArray(Ba);
            byte[] ba = new byte[Ba.length];
            for (int i = 0; i < ba.length; i++) {
                  ba[i] = Ba[i].byteValue();
            }
            ret = new Address (ba);
      return (ret);
   }

      /**
```

```
	 * @param address :
	 * @return this <= address
	 */
	public boolean leq(Address address) {
		if (address.address.length > this.address.length)
			return true;
		if (address.address.length < this.address.length)
			return false;
		for (int i = 0; i < address.address.length; i++) {
			if ((address.address[i] & 0xFF) < (this.address[i] & 0xFF))
				return false;
			if ((address.address[i] & 0xFF) > (this.address[i] & 0xFF))
				return true;
		}
		return true;
	}

	/**
	 * @param address
	 * @return this < address
	 */
	public boolean le(Address address) {
		if (address.address.length > this.address.length)
			return true;
		if (address.address.length < this.address.length)
			return false;
		for (int i = 0; i < address.address.length; i++) {
			if ((address.address[i] & 0xFF) < (this.address[i] & 0xFF))
				return false;
			if ((address.address[i] & 0xFF) > (this.address[i] & 0xFF))
				return true;
		}
		return false;
	}

	/* (non-Javadoc)
	 * @see java.lang.Object#toString()
	 */
	public String toString() {
		return(Tools.ba2s(getAddress()));
	}
}
```

Bibliographie

Guennadiev Boris, « **La carte à puce, l'API Java Card, le Button DS1957B »,** Examen probatoire du 12 janvier 2006, proposé par Pierre Paradinas, CNAM.

Pierre Paradinas, Support de cours sur « **Java Card** », UV de Systèmes Enfouis et Embarqués, Valeur C, Laboratoire CEDRIC, CNAM. Accessible via :
http://deptinfo.cnam.fr/~paradinas/cours/ValC-IntroJavaCard.pdf

Wolfgang Rankl and Wolfgang Effing**, « Smart Card Handbook**», 3rd Edition, John Wiley & Sons Ed., 2003, ISBN 0-470-85668-8.