Soutenance de Projet Professionnel

# Configuration d'une Application d'Empreintes biométriques sur Carte Java


Présenté par:

**Mekki CHRIGUI**


Projet réalisé en collaboration avec:


&

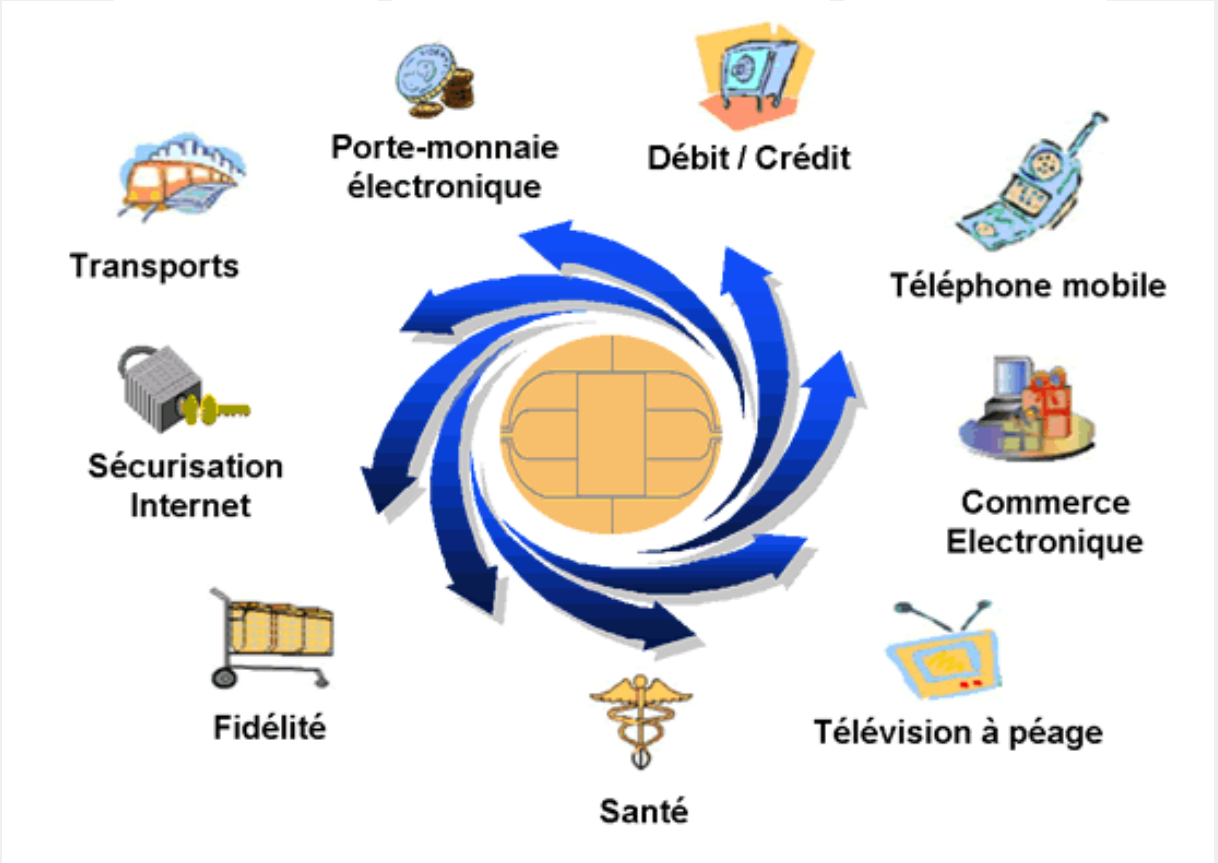

Tunis, 03 juillet 2002



# PLAN

- Cartes à puce : Etat de l ’art
- La technologie carte à puce Java
- La spécification Java Card par rapport à Java
- Conception d’une applet carte
- Construction de l’application client
- Présentation du système RAD
- Mise en œuvre de l’application
- Conclusions et perspectives.

# Les applications courantes de la carte

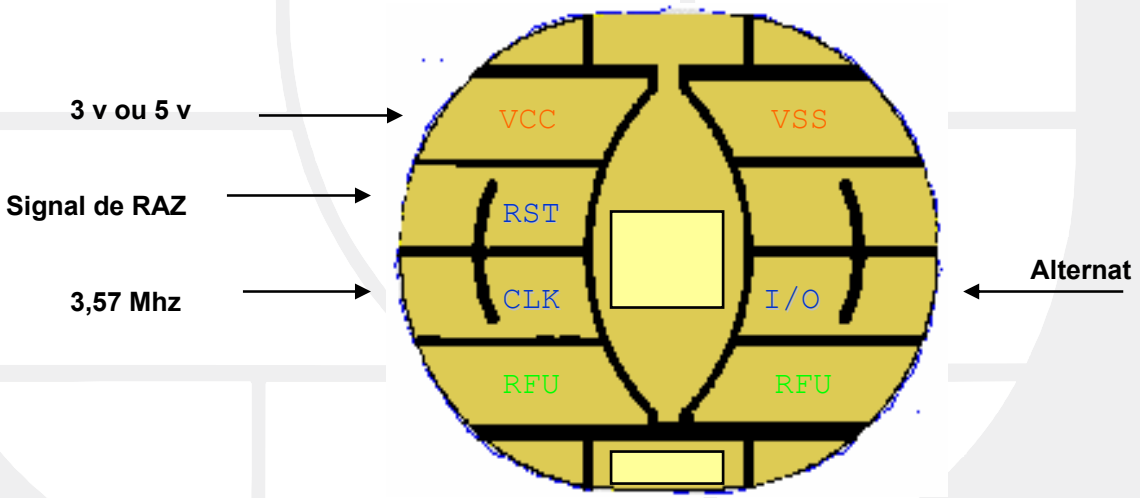


# Terminologie de base

- **Carte Magnétique (Magnetic stripe card)**
  - Piste magnétique au verso
  - Carte de fidélité
- **Carte à mémoire (Memory card)**
  - Mémoire EEPROM
  - Télécartes (1982), fidélité
- **Carte à microprocesseur (Microprocessor card)**
  - μP, RAM, ROM. Microcontrôleur encarté.
  - Exemple : carte bancaire (1985)

# Normalisation du matériel

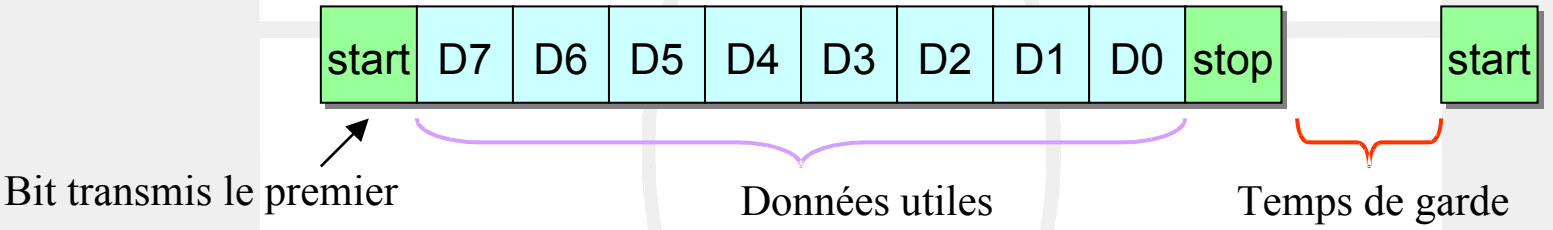
- **ISO 7816**
  - Partie 1 : Caractéristiques physiques
  - Partie 2 : Dimensions et positions des contacts
    - Format carte de crédit (85 * 54 * 0.76 mm)
    - Définition des contraintes que doit supporter la carte




  - Partie 3 : Caractéristiques électriques

# Normalisation de la communication

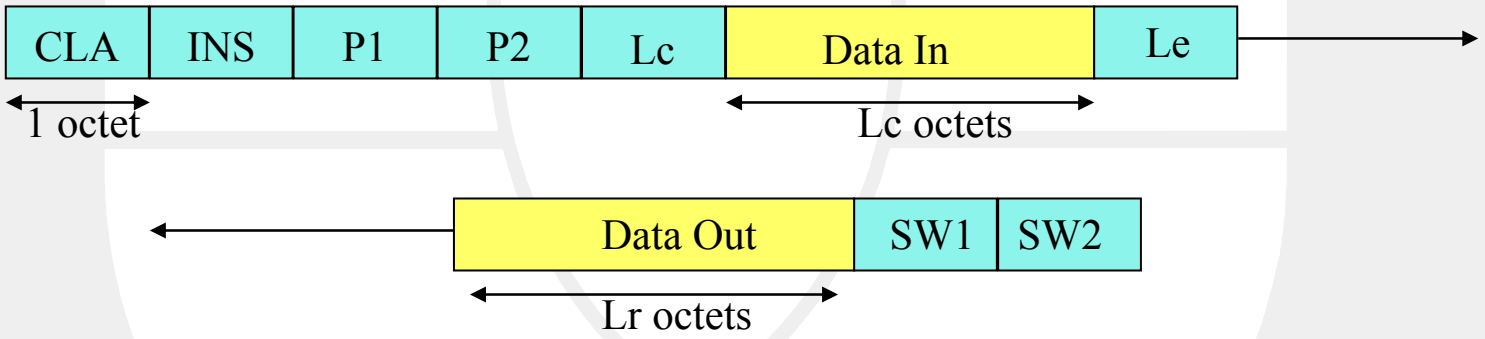
- **ISO 7816 - 3 pour protocoles lecteur-carte**
  - Transmission d'un caractère
    - 1 bit de démarrage, 8 bits de données, 1 bit de parité
    - Définition d 'un temps de garde entre 2 caractères




  - Réponse de la carte à la Remise à Zéro (ATR, Answer to Reset) :
    - Séquence d 'octets décrivant les caractéristiques de la carte
  - Protocoles de communication (asynchrones et semi-duplex)
    - Mode Maître-esclave : La carte répond à des commandes
    - T=0 : Transmission de caractères (le plus utilisé)
    - T=1 : Transmission de blocs de caractères

# Normalisation du format des commandes

- **ISO 7816-4**
  - Définition du format des paquets de données échangés entre un lecteur et une carte :
    - APDUs (Application Programming Data Units) de commande et de réponse




  - Mots d'état de réponse SW1 et SW2 standardisés (OK = 90 00 en hexadécimal)

# Microcontrôleur pour carte

- **Fondé sur la technologie MAM**
  - Microprocesseur + bus + mémoires réunis sur un même substrat de silicium
- **Types de microprocesseurs**
  - 8 bits (CISC) - 32 (RISC) bits
  - STM, Philips, Siemens, Motorola, Infineon, Atmel, Nec
- **Types de mémoires**
  - ROM (Read Only Memory) : Mémoire non volatile à lecture seule (---> 128 Ko)
  - RAM (Random Access Memory ): Mémoire volatile à accès rapide (---> 2 Ko)
  - EEPROM (Electrical Erasable Programmable ROM) : Mémoire non volatile réinscriptible (---> 64 Ko)

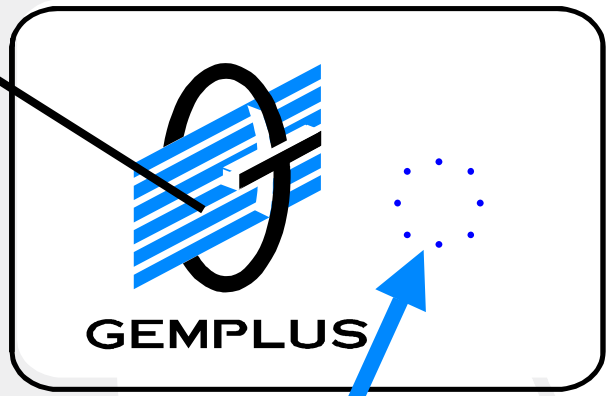
# Cycle de vie d'une carte (1/3)

- **Fabrication**
  - Inscription d'un programme en mémoire ROM définissant les fonctionnalités de base de la carte :
    - Masque figé capable de traiter un nombre limité de commandes prédéfinies
- **Initialisation**
  - Inscription en EEPROM des données communes à l'application
- **Personnalisation**
  - Inscription en EEPROM des données relatives à chaque porteur

# nitialisation / Personalisation / Emissior

- Initialisation

Logo

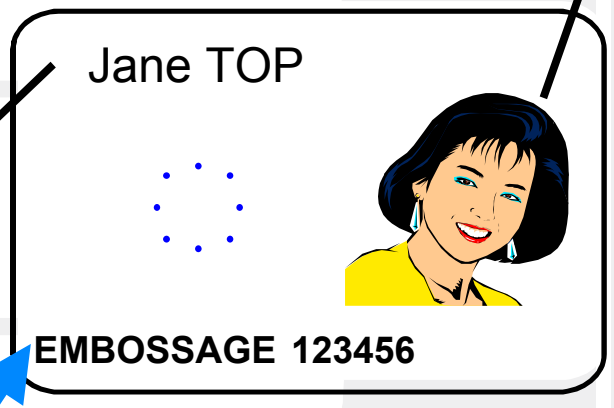



Architecture commune du circuit intégré (puce)

- Personnalisation

Photo

Individual information



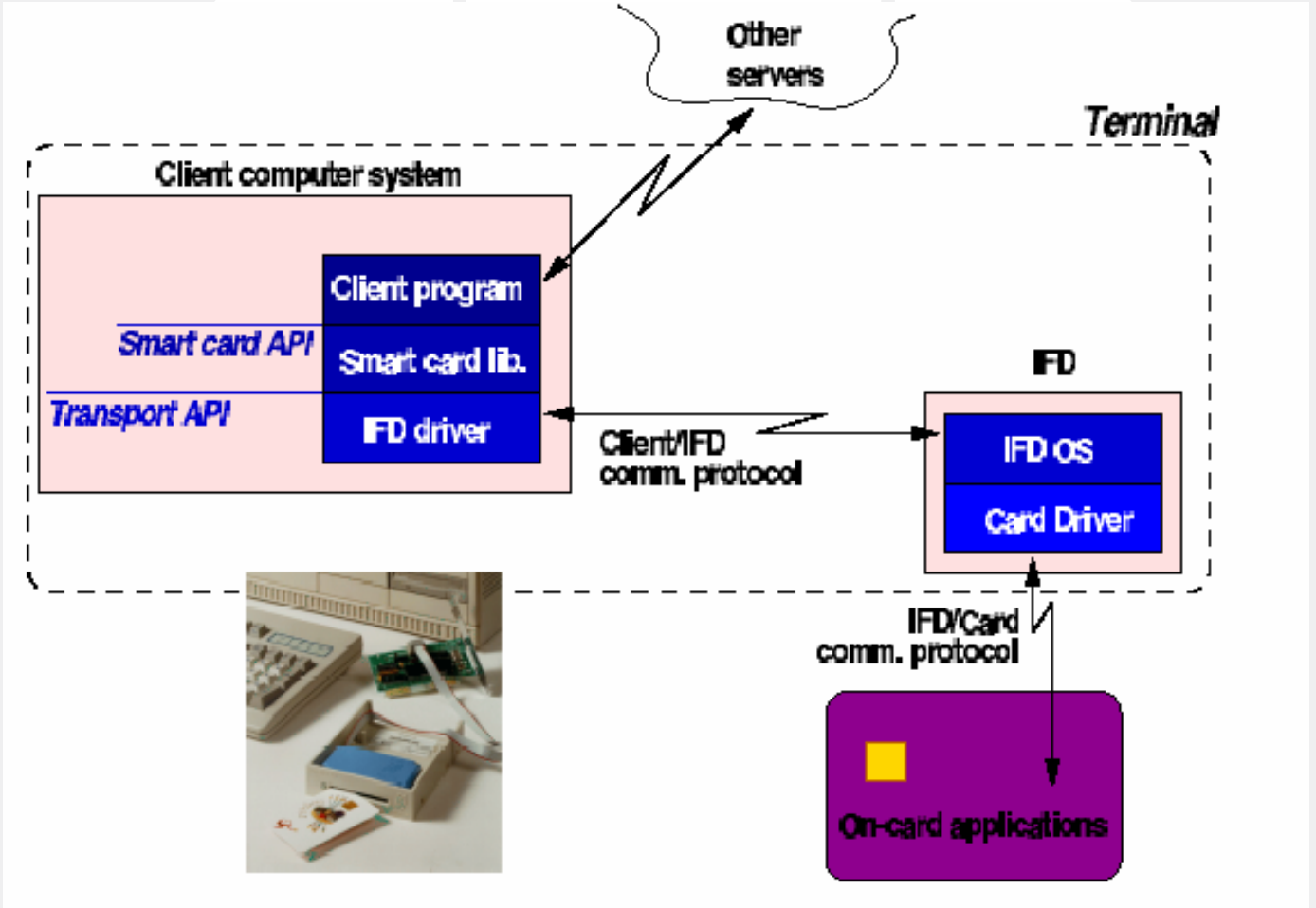

Information Individuelle

Numéro unique imprimé et enregistré dans la puce

# Cycle de vie d ’une carte (3/3)

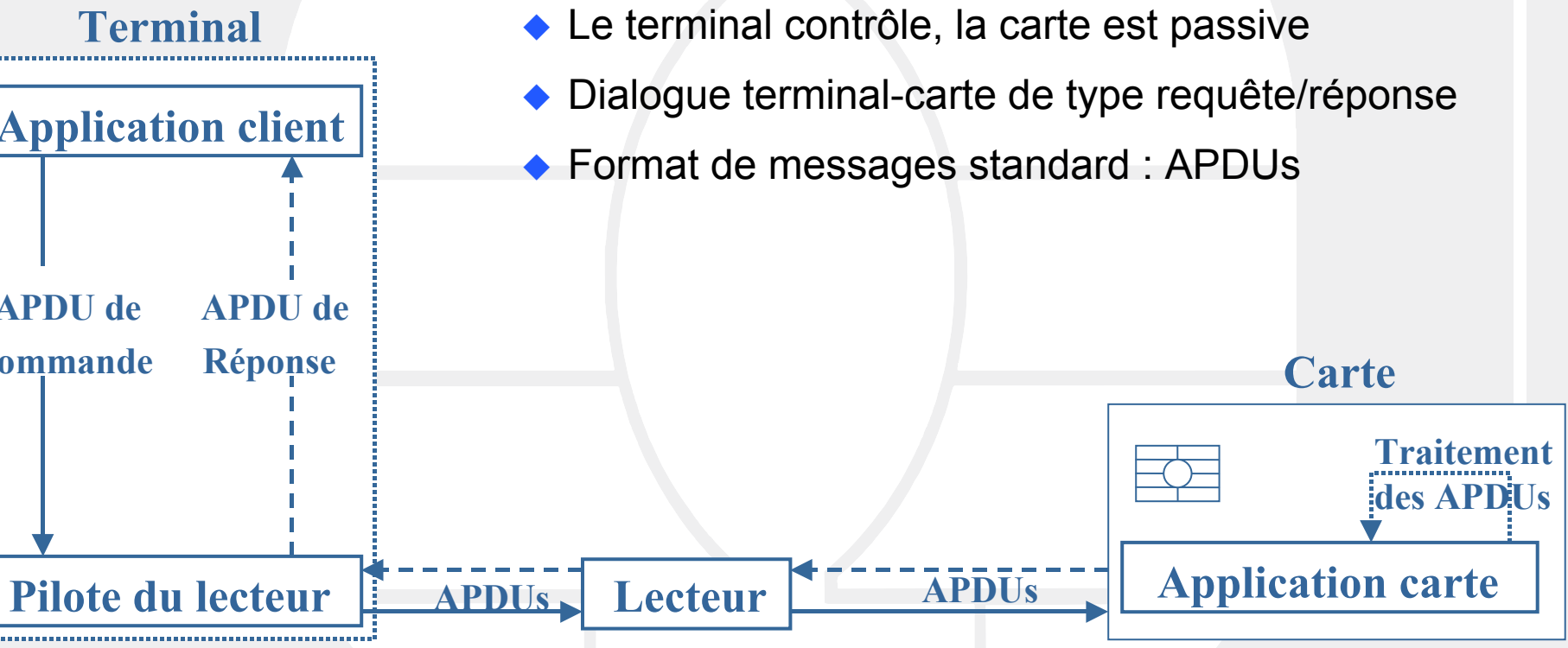
- **Utilisation**
  - Envoi d ’APDUs de commande à la carte
  - Traitement de ces commandes par le masque de la carte
    - Si commande reconnue
      - Traitement en interne de la commande ⟹ lecture/écriture de données en EEPROM
      - Renvoi d ’un APDU de réponse
    - Si commande inconnue
      - Renvoi d’un code d’erreur
- **Fin**
  - Par invalidation logique, saturation de la mémoire, bris, perte, vol, etc

# Interface de la carte à puce



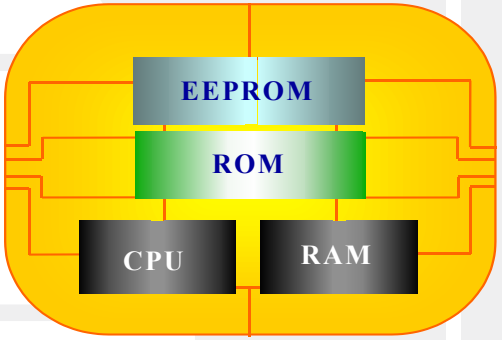
# Architecture d ’application carte

- **Schéma général**
  - Le terminal contrôle, la carte est passive
  - Dialogue terminal-carte de type requête/réponse
  - Format de messages standard : APDUs



# Une carte type

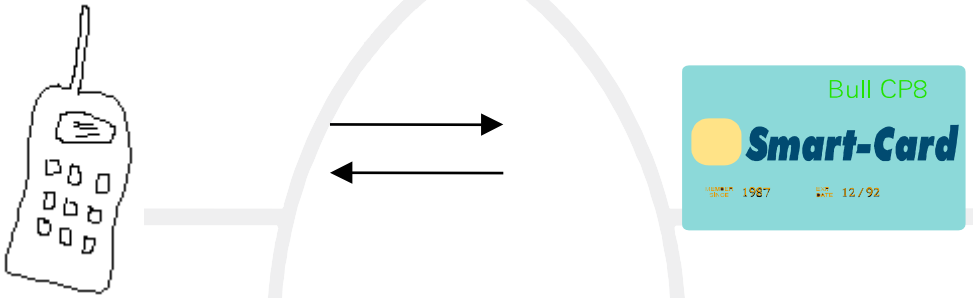
- Microprocesseur (CPU) 8 bits @ 4 MHZ
  - ROM lecture seule : 8 - 32 kilo-octets
  - RAM lecture-écriture : 256-1024 kilo-octets
  - EEPROM réinscriptible : 2-32 kilo-octets




- Interface de communication half-duplex @ 9600 bits/s
- Programmation au niveau processeur en assembleur. Code logé en ROM.
- Communication par APDU

# Exemple d ’application : carte SIM

- **Terminal = Client** **Carte = serveur**



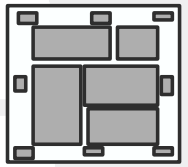

- **Fonctions de la carte SIM :**
  - Authentification de l ’opérateur et du porteur
  - Accès au réseau
  - Génération des clés de chiffrement de la communication
  - Stockage de répertoires, de messages SMS
  - Services divers

# Résumé de l'état de l'art

- La carte à puce est un véritable ordinateur utilisé comme serveur portable et sécurisé de données personnelles
- Elle était programmée comme un composant embarqué avec un code applicatif figé
- Elle était difficile à intégrer dans les systèmes d'information

# Facteurs de changement

- **Evolution technologique**
  - Densité d ’intégration liée à la finesse de la gravure : 0,8 --> 0,18
    - Intégration de processeur RISC 32 bits
    - Augmentation des capacités mémoire
    - Augmentation de la fréquence d’horloge
  - Apparition des technologies sans contact
- **Demande des opérateurs**
  - Téléchargement d ’applications
  - Multi-applicatif
- **Demande des utilisateurs**
  - Simplicité d’utilisation



# Vers des cartes plus ouvertes :

- **Problèmes à résoudre et/ou besoins à satisfaire :**
  - Permettre le développement de programmes pour la carte sans avoir besoin de graver un nouveau masque
    - Faciliter les développements de code dans la carte
  - Faire de la carte un environnement d'exécution de programmes ouvert
    - Rendre plus souples et plus évolutives les applications cartes
  - Faciliter le développement d 'applications clientes

# Eléments de solutions :

- **La carte Java** :

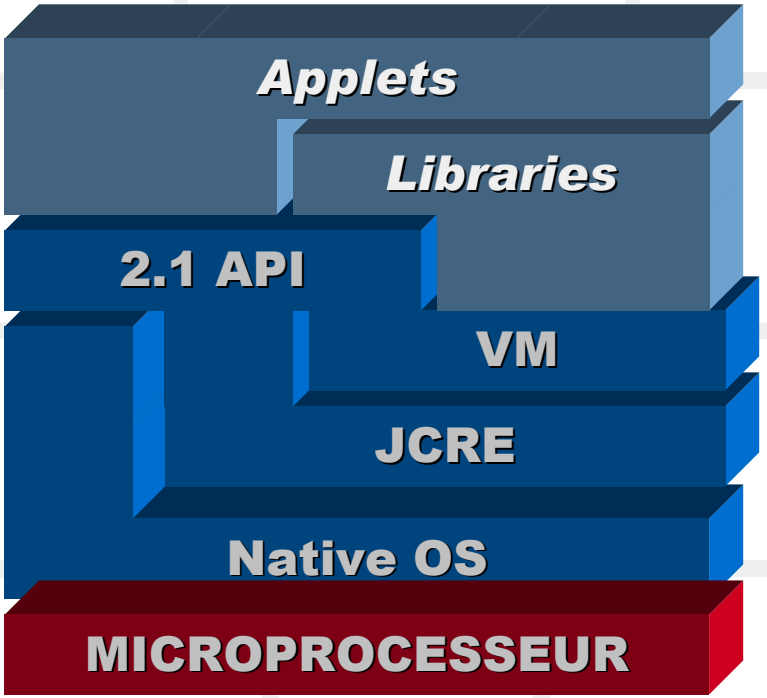


  - Utilise le langage Java pour programmer les cartes (Bénéfice d'un langage orienté objet)
  - Utilise la plate-forme Java GemXpresso pour charger des applets cartes (Bénéfice d'une architecture sécuritaire)
  - Mise en œuvre des spécifications Open Platform pour sécuriser les échanges avec la carte.

# Que représente Javacard 2.1

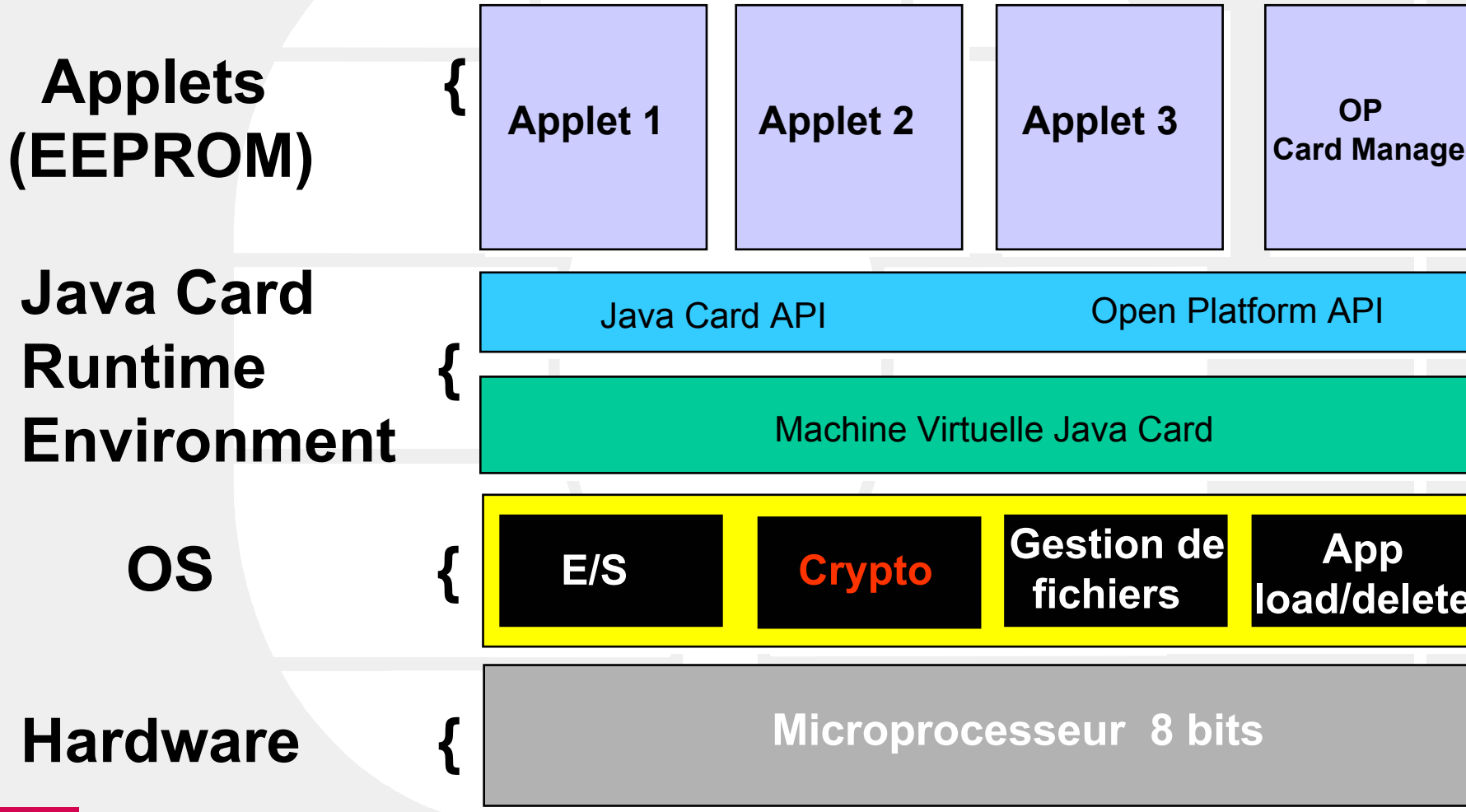
- **Un jeu de spécifications**
  - Créées par Sun Microsystèmes
  - Basées sur le language Java
- **Trois parties**
  - Application Programming Interface (API)
  - Java Card Run Time Environment
  - Java Card Virtual Machine (Machine Virtuelle)

# Architecture de la Javacard 2.1




Le Système d ’exploitation (OS) pilote un microcontrôleur 8 bits doté de 32 Ko de ROM, 32 Ko d ’EEPROM et 2 Ko de RAM.
**20 Ko d ’EEPROM et 1,5 Ko de RAM sont disponibles pour les Appli cations Javacard (Applets cartes) développées**

# Architecture de la carte GemXpresso 211



# Qu'est-ce qu'une Java Card

- **Une Java Card est une carte à puce qui peut exécuter des programmes Java (Applets carte)**
  - La spécification Java card 2.1 définit un sous-ensemble de Java dédié à la carte à puce :
    - Sous ensemble du langage Java (à cause des limitations carte : composant 8 bits, 512 octets RAM, 32 KO EEPROM, 24 KO ROM)
    - Sous ensemble du paquetage Java.lang
    - Découpage de la machine virtuelle Java
    - API spécifique à la carte

# L'API Java Card

- Les classes de l 'API Java Card définissent les fonctionnalités fournies aux programmeurs. L'API définit les conventions d'interactions du JCRE avec l 'applet
- L'API contient 4 paquetages :
  - java.lang (lié à la machine virtuelle)
  - java.framework (lié de près au protocole carte)
  - javacard.security
  - javacardx.crypto

# Le langage Javacard

- **Supporte**
  - boolean, byte, short
  - int (optionally)
  - Objects
  - Arrays
  - Virtual methods
  - Dynamic allocation
  - Packages
  - Exceptions
  - Interfaces
- **Ne supporte pas**
  - float, double, long
  - char, String
  - Multi-dimensional arrays
  - Garbage collection
  - Finalization
  - Threads
  - Class loader
  - Security manager

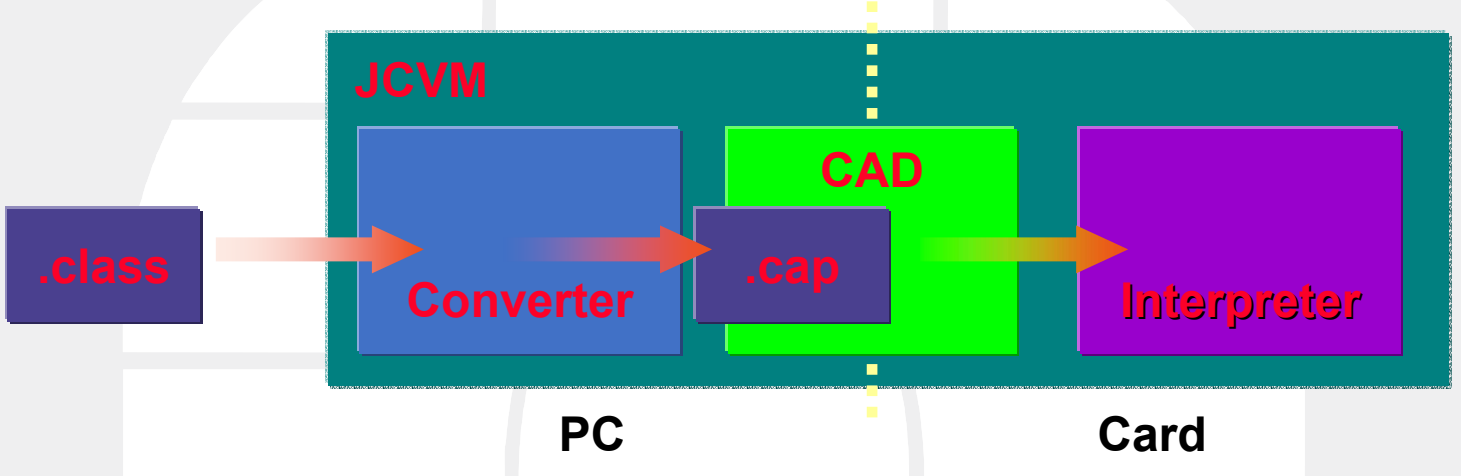
# Machine virtuelle JC : partie carte

- Définition de la machine virtuelle Java
  - Vérifieur de bytecode
  - Chargeur dynamique de classes
  - Interprêteur de bytecode
- Dans la carte la Java Card VM n'implémente que :
  - Interpréteur de byte-code
  - Gestion des classes et objets
  - Isolation des applets
    - Par paquetage

# Machine virtuelle JC : partie hors carte

- Vérifieur de bytecode
  - Utilise le vérifieur de bytecode Java classique
  - Contrôle le sous ensemble Javacard (langage + API)
- Convertisseur
  - Initialise les structures de données de la JCVM
  - Optimisation mémoire

# Architecture modulaire de la Machine Virtuelle

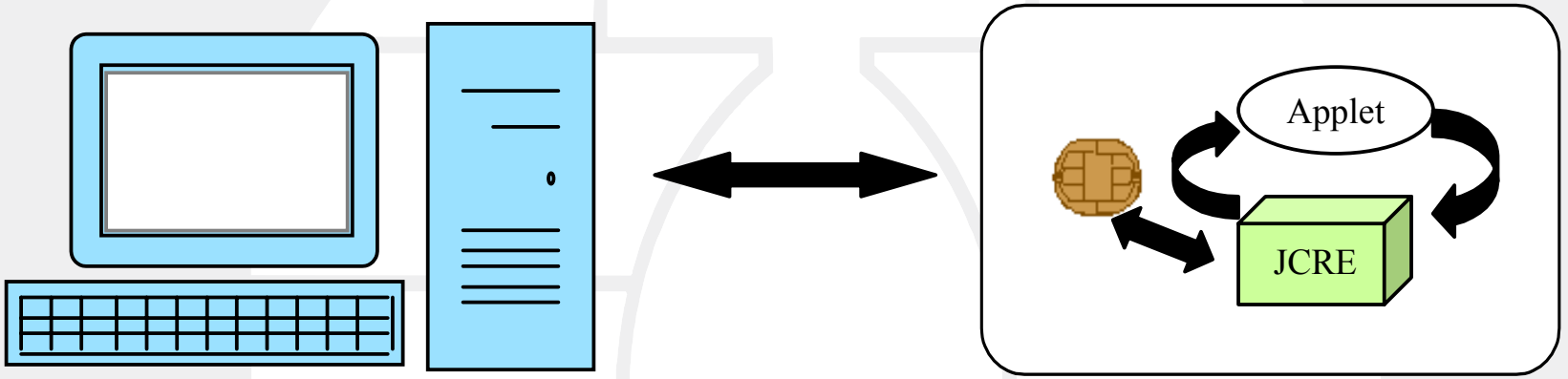



- Le convertisseur (off-card)
  - Chargeur de classes, chaînage et résolution des noms
  - Vérification
  - Optimisation des Bytecode et conversion
- L'interpréteur (on-card)
  - Exécution des Bytecodes et respect de la sécurité

# Le JCRE

- Il définit comment une carte Java gère ses ressources
- Il définit les contraintes sur l'OS Java Card
  - Comment le problème de l'atomicité est résolu
  - Comment la communication est pilotée
  - Comment les applications sont gérées
  - Comment les mesures de sécurité sont appliquées
- Il représente le cœur d'une carte Java

# Le JCRE et les APDUs




Le JCRE filtre le flux d ’APDUs entre les applets installées et le terminal lecteur :
Toutes les APDUs sont reçues par le JCRE qui passe une instance de la classe APDU à la méthode process de l’applet active.

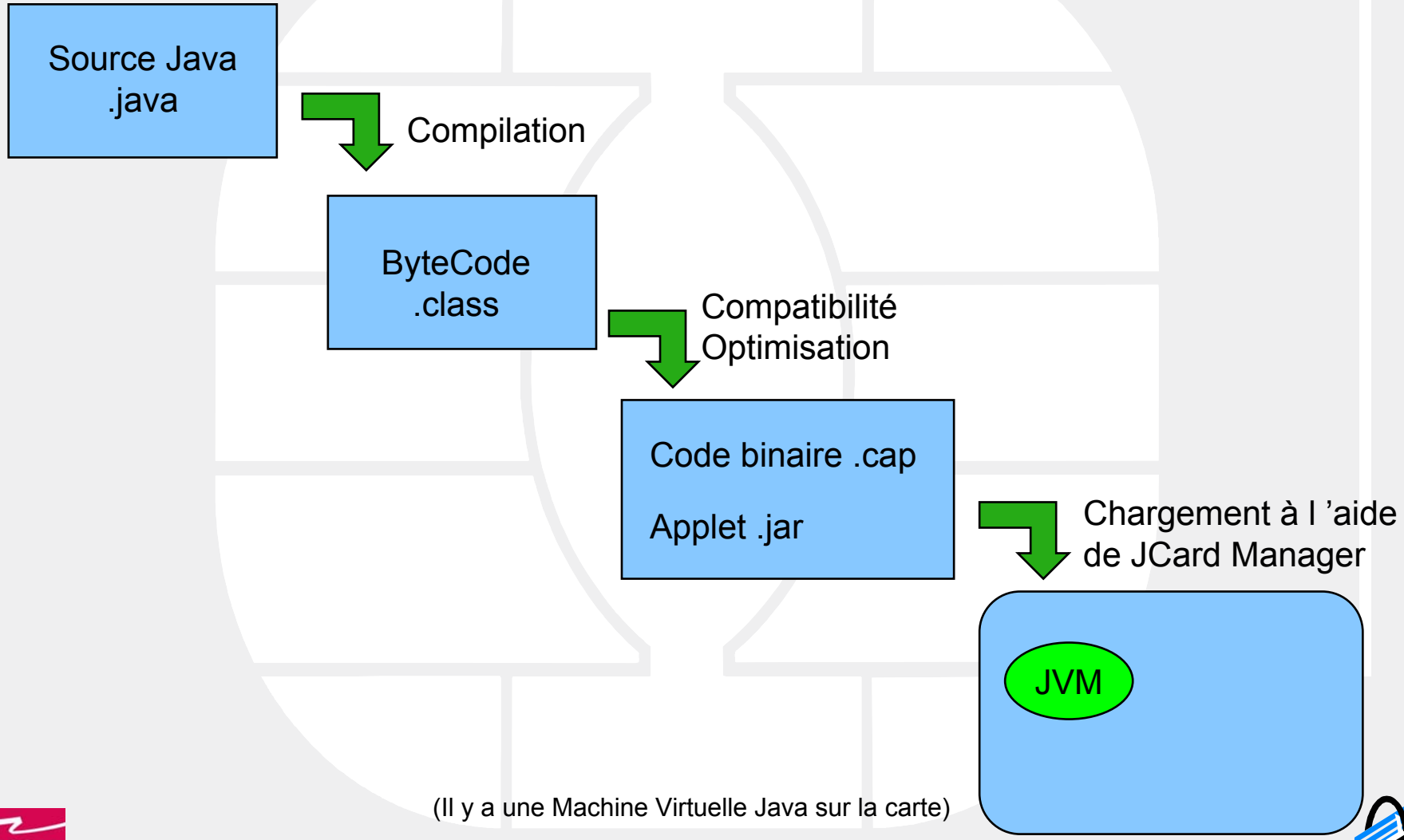
# Ouvrir un projet : xxx.jpr

- Lancer JBuilder4
- Ouvrir le projet (menu Fichier) dans :

C:\Mes documents\V3\.jbuilder4\myproject\myproject.jpr

- Revérifier les propriétés du projet (menu Projet)
- Constuire le projet (menu Projet)
- Convertir le Projet (menu GemXpressoRad)
- Lancer le JCardManager

# Le développement (2) : applet




(Il y a une Machine Virtuelle Java sur la carte)

# Précisions sur l'applet carte

- Une applet carte est un programme serveur de la javacard
  - réagit à une APDU de sélection (du terminal)
  - sélectionné par un identifiant AID (ISO 7816-5)
- Une fois installée dans la carte, est toujours disponible
- Classe qui hérite de **Javacard.framework.Applet.**
  - **Cette classe est la superclasse des applets carte**
- Doit implémenter les méthodes qui interagissent avec le JCRE :
  - install(), select(), deselect(), process()

# Méthodes publiques d'une applet

- public static void install (APDU apdu)
  - Appelée par le JCRE quand l 'applet est chargée dans la carte **pour créer l'instance de l 'applet**
- public boolean select ()
  - Appelée par le JCRE quand une APDU de sélection est reçue et désigne cette applet
  - **Rend l'applet active**
- public void process (APDU apdu)
  - Appelée par le JCRE quand une APDU de commande est reçue pour l 'applet active
- public void deselect ()

# Gestion des APDUs par une applet

- L'unité de traitement de base d'une applet est un objet de type javacard.framework.APDU
  - Transmis par le JCRE à la réception d 'une APDU de commande de la carte
    - Appel à la méthode process () de l 'applet courante
- Classe javacard.framework.APDU
  - Encapsule les échanges de message APDUs (commandes et réponses) dans un buffer d 'entrées/sorties de 261 octets

# Développement Javacard (1)

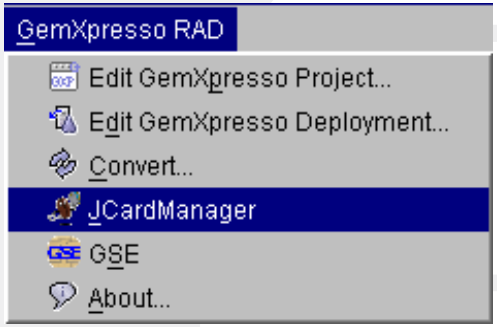
## Deux types d ’applications à développer en parallèle :

- **Application Client**
  - Développée en Java (Environnement Visual Cafe, JBuilder)
  - Echange les APDUs avec l’application serveur
- **Application serveur**
  - Applet Java (.jar) à installer et rendre active (select)
  - Applets traitent les APDU de commande envoyées par l ’application cliente

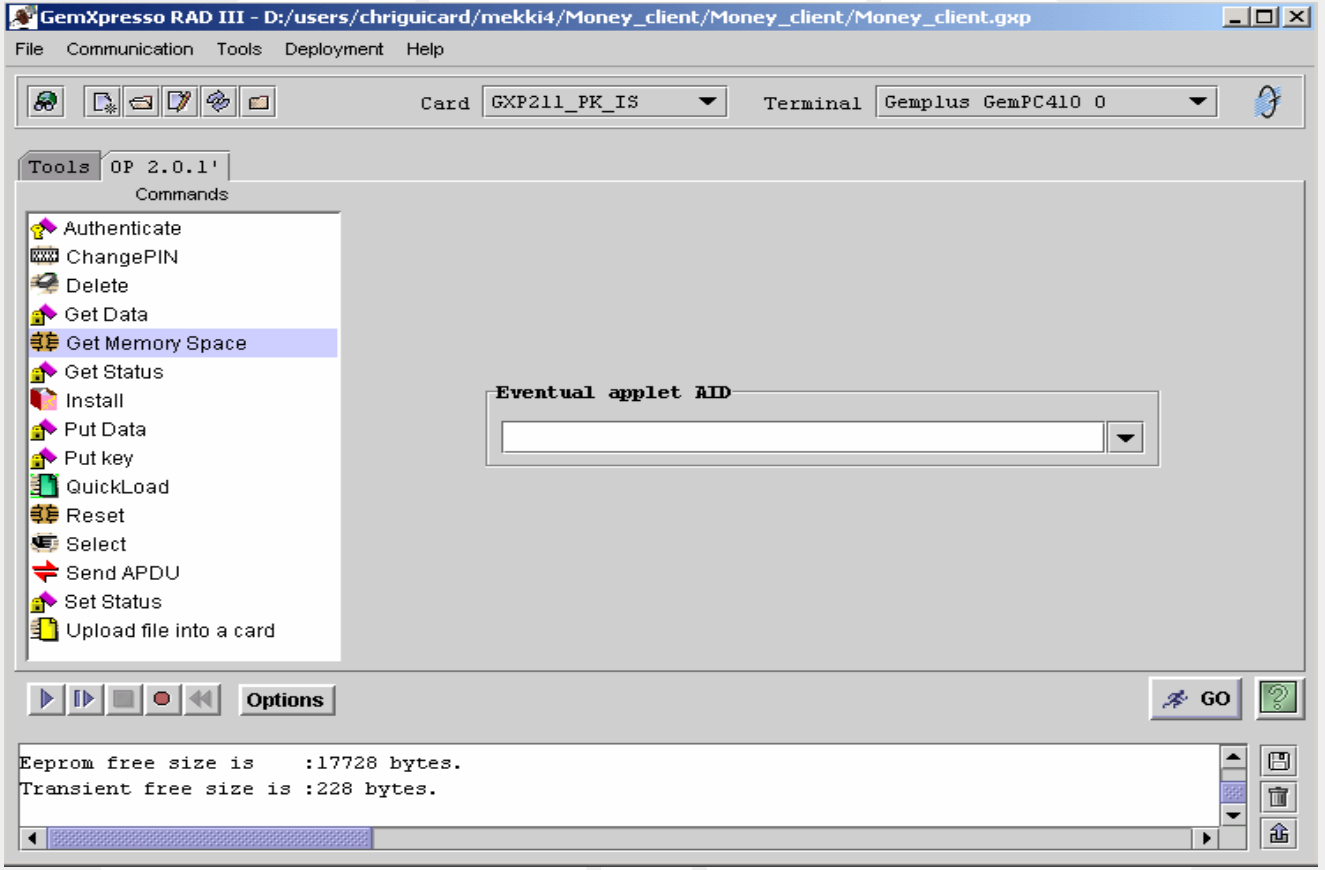
# Un canal de communication sécurisé

- Grâce aux **spécifications Open Platform**, mise en œuvre par Visa dans l ’implémentation VOP, on dispose d’une norme efficace de sécurisation du lien entre la carte et le terminal à chaque session :
  - Mise en œuvre d’un canal sécurisé
  - Celui-ci permet d’implémenter :
    - Une authentification mutuelle : (mêmes clés)
    - La confidentialité : cryptage données (3DES)
    - L’intégrité : calcul, vérification de Macs (3DES)

# Lancement du Jcard Manager




# Utilisation et fonctions du Jcard Manager(1)

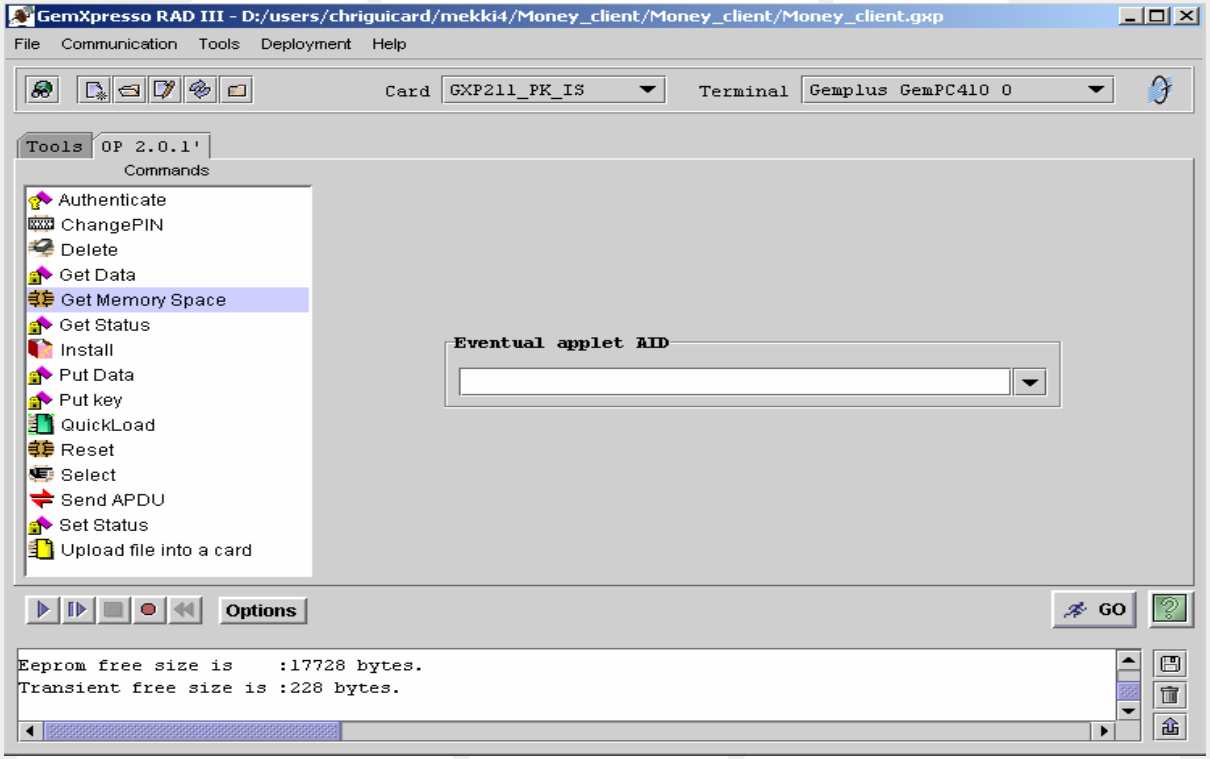



- Application cliente générique
- Permet d ’accéder à la carte insérée dans un lecteur

# Utilisation et fonctions du Jcard Manager(2)

- Permet d'effacer les applets précédentes
- Permet de tester une applet carte sans application client à l'aide d 'envoi d 'APDU
- Opérations possibles : Reset, Authenticate, Install, Select, Upload jarfiles, Delete
- Téléchargement des applets
- L'Installation termine la phase de chargement
- Installation et Sélection d 'une applet
- Envoi d 'APDUs après sélection de l'applet

# Vérification de l'espace mémoire disponible




- 20 Ko d 'EEPROM et 1,5 Ko de RAM sont disponibles pour les Applications Javacard (Applets cartes) développées
- Théoriquement c'est 20 ko mais réellement c'est 17728 octets

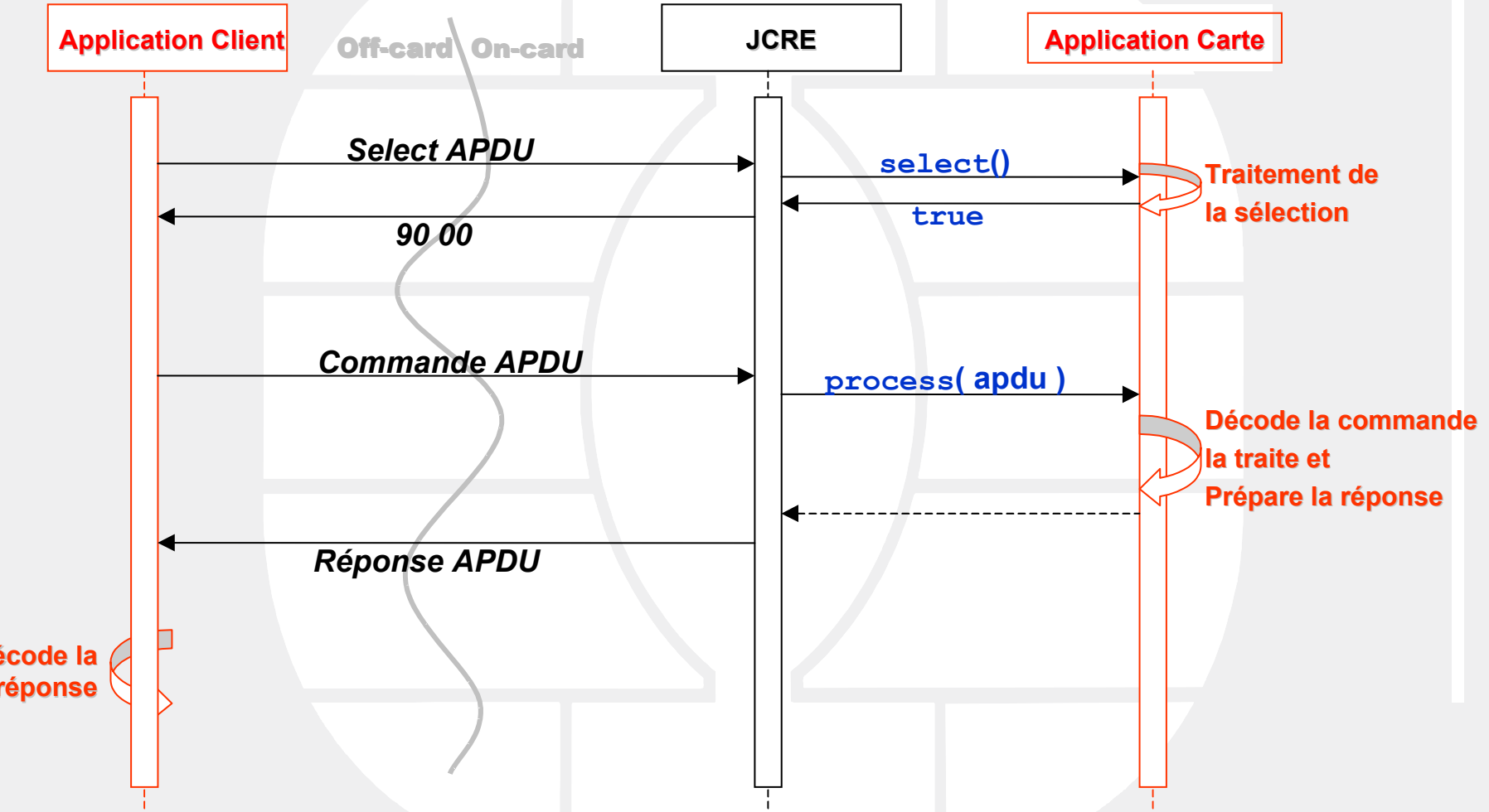
# Qu'est-ce qu'un APDU (Application Protocol Data Unit)

- C'est la communication entre le lecteur et la carte dont la structure est définit par le standard ISO7816.
- Deux catégories d'APDUs:commandes APDUs et réponse APDUs.
- Le JCRE reçoit les données du CAD et les charge dans le buffer de données d'APDU.
- Envoyées du lecteur vers la carte, les commandes APDU contiennent une entête obligatoire 5-bytes et des données de 0 à 255 bytes.
- De la carte vers le lecteur, les réponses APDUs deux status word obligatoire 2-bytes et des données de 0 à 255 bytes.

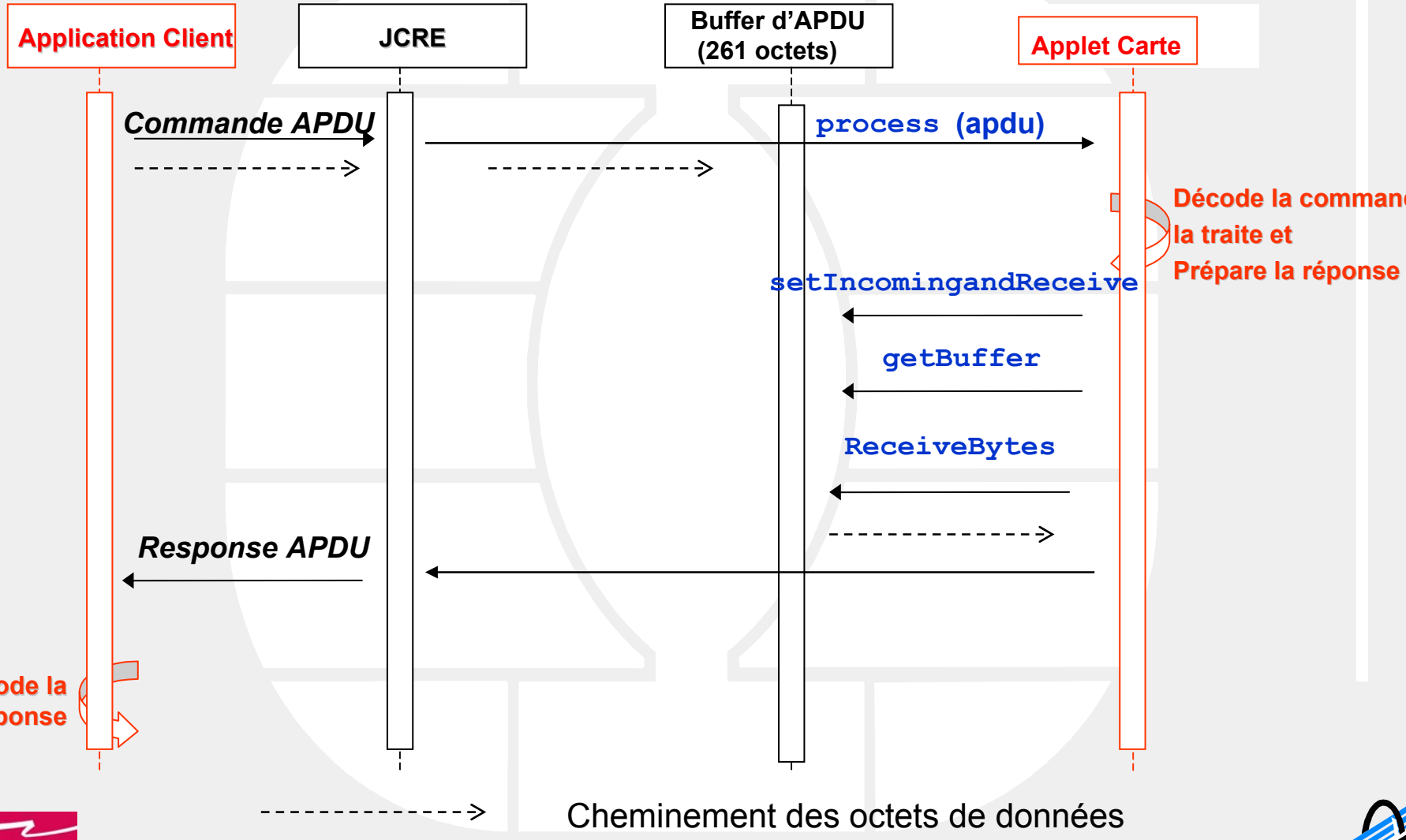
# Stockage d'empreinte sur la carte

- La taille des informations(champ de caractères, photo,empreinte)et des applets ne doit pas dépasser 17ko, pour pouvoir être téléchargés.
- Les photos ou empreintes à stocker doivent être numérisés (code binaire).
- Le buffer de taille maximale 261 octets, ne peut pas échanger des données de quelques kilos octets en une seule transaction.
- Une photo(.GIF ou .JPEG)de quelques koctets doit être découper en des paquets de 255 octets ou moins.

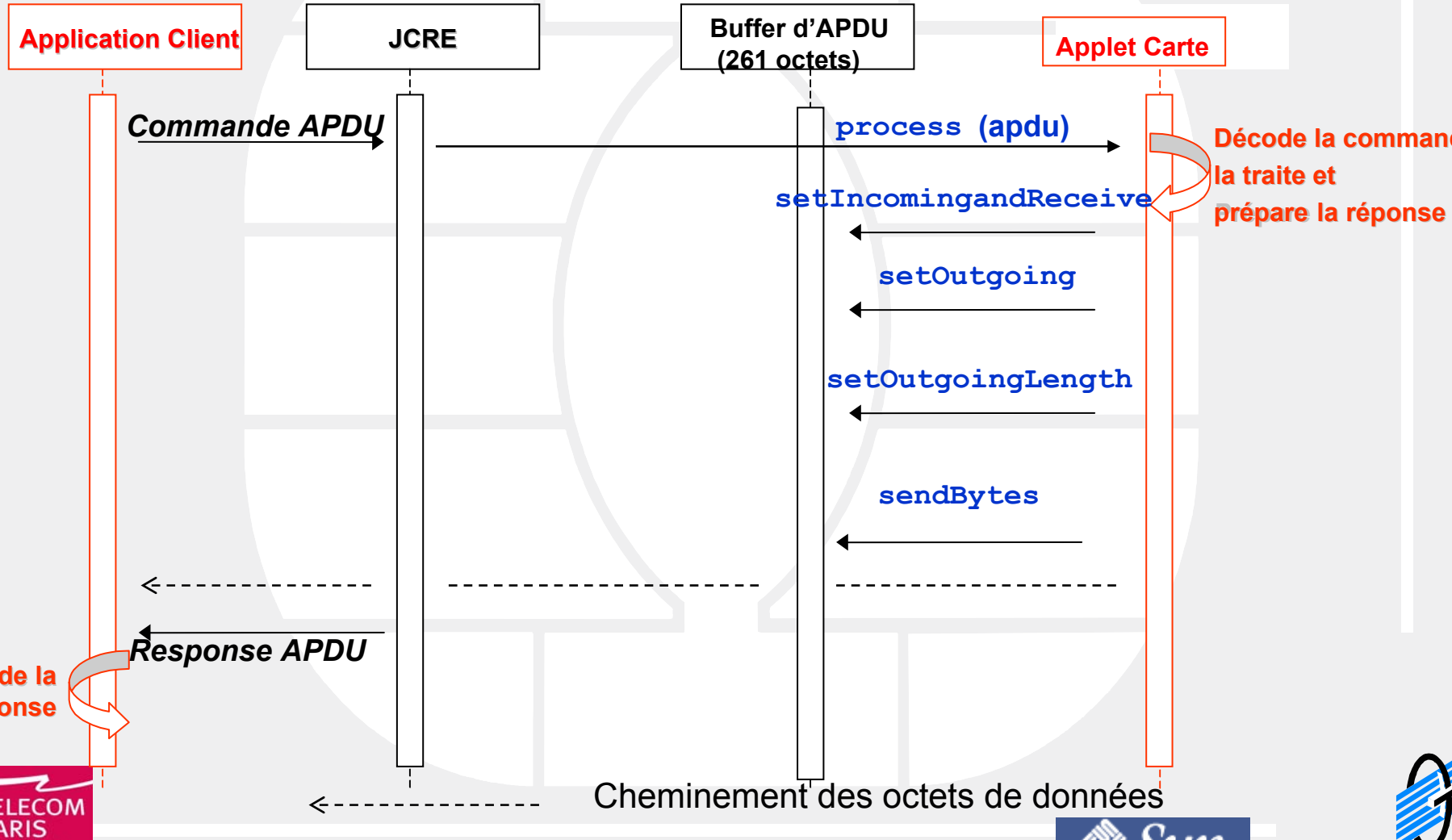
# Javacard : Séquencement



# Télécharge Photo : Gestion du buffer



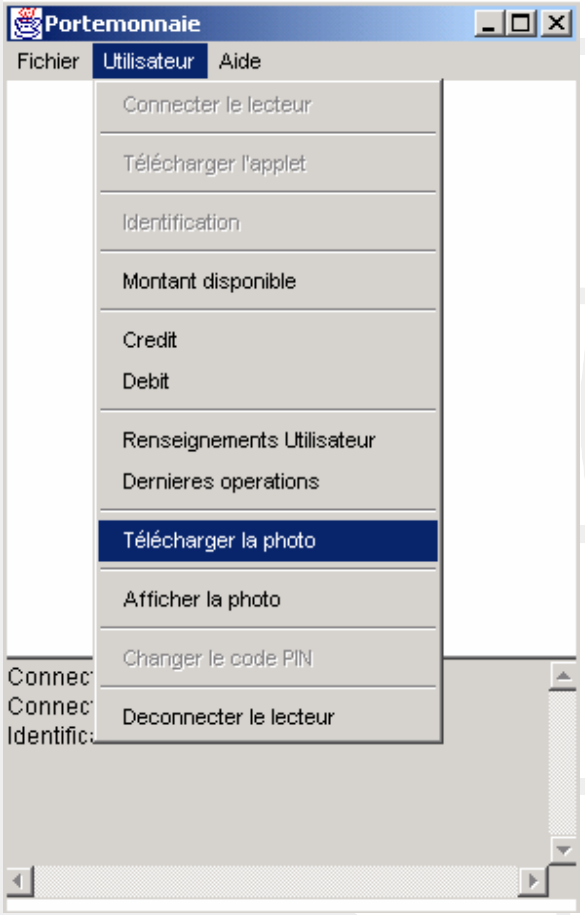
# Affiche photo : Gestion du Buffer

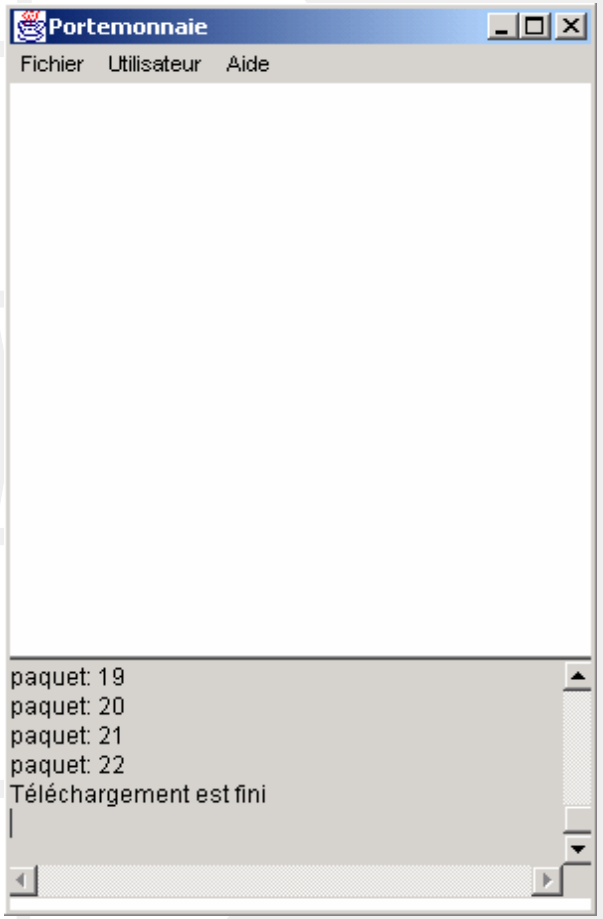



Cheminement des octets de données

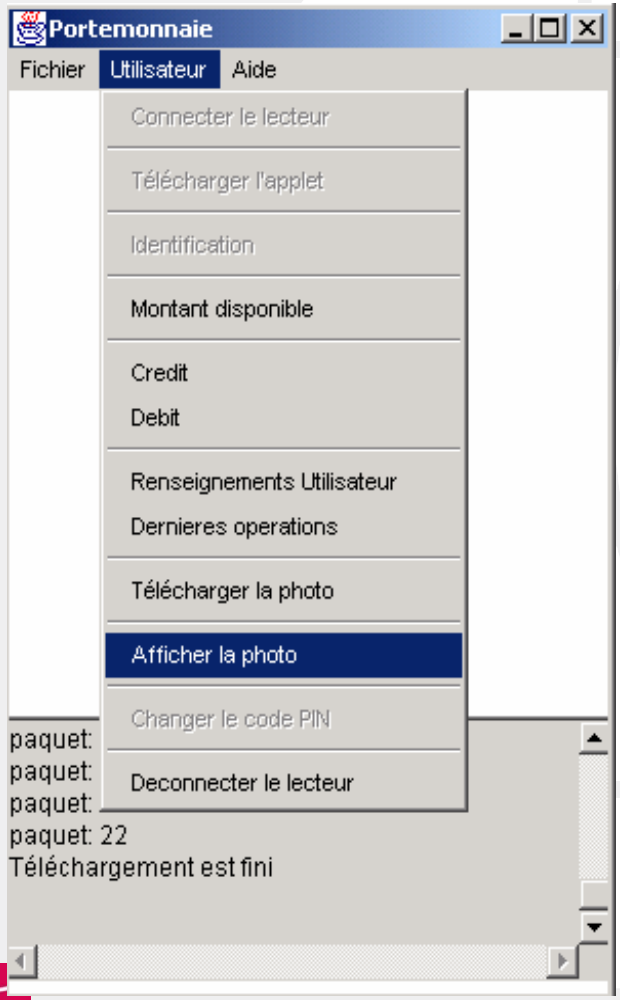
# Carte à puce Java et stockage de données bio métriques

# Téléchargement de la photo



# Affichage de la photo




# Conclusions sur Java Card

- Java Card is Java …
  - Un large sous ensemble du langage est supporté
  - La conception orientée objet peut être utilisée dans les programmes
- … but it is Java on a smart card
  - La gestion Mémoire est un peu difficile
  - La taille du code est un objectif majeur
  - La vitesse d'exécution est lente

# *Remerciements*

*Je remercie vivement*

*Monsieur JEAN LEROUX LES JARDINS, Directeur départemental adjoint des Télécommunications, qui a dirigé ce projet et qui m'a fait l'honneur de s'intéresser à ce travail, pour la disponibilité qu'il a toujours manifestée à mon égard.*

*Je remercie également Monsieur AHMED CHERIF, mon encadreur à la STEG pour son chaleureux accueil et sa collaboration dans la réalisation de ce travail.*

*Tous mes remerciements s'adressent aussi à Monsieur FETHI TLILI, mon encadreur à Sup'Com pour ses conseils et son aide fournis durant ce stage.*

*Finalement, Je tient à exprimer ma gratitude aux responsables Sup'Com-STEG pour leur grande contribution à la réussite de ce cycle de DESS.*
*Veuillez messieurs accepter l'expression de mes remerciements les plus sincères.*

# LISTE DES FIGURES

# Introduction générale

## Présentation du projet

L'objet de ce rapport est l'étude de l'ingénierie logicielle permettant le développement d'applications de carte à puce par le billet de carte java. Il est destiné aux personnes qui désirent étudier les éléments essentiels de la technologie des cartes à puce et de la programmation des cartes java. Il traite et répond aux problématiques suivantes: les particularités spécifiques des cartes à puce, comment les applications de la carte à puce peuvent être programmées avec la carte java, et l'utilisation des techniques réparties dans le but de perfectionner la programmation des applications de carte à puce. Au cours de ce rapport, on se concentre sur les méthodes et les outils impliqués pendant le processus de développement d'une application de la carte à puce.

Ce rapport est organisé en cinq chapitres:

- Chapitre1, ''Généralités sur la carte à puce '', fournit une présentation générale du domaine spécifique de la carte à puce et des motivations qui ont mené à l'état de l'art actuel de la programmation de ces cartes.
- Chapitre2, ''La technologie de la carte de java'' fournit un premier aperçu à la technologie de la carte java et aux techniques de programmation répartie qui peuvent être appliqués aux cartes à puce.
- Chapitre3, ''Développement d'un applet pour la carte de Java'', décrit au fond comment développer une application avec la carte java.
- Chpitre4, ''A propos de la biométrie'', donne un aperçu sur les concepts biométriques et les techniques d'assortiment d'une empreinte digitale.
- Chapitre5, ''Environnement de développement de carte à puce Java'', permet de se familiariser avec l'environnement de développement des cartes à puce java à travers l'explication et l'exécution de quelques applications.

En effet ; mon travail avait pour but de réaliser une étude préliminaire sur les possibilités de stockage offertes par les cartes à puce Java. Le langage Java est en passe de devenir le langage de prédilection pour les applications destinées à beaucoup d'objets mobiles car il présente la

particularité de pouvoir exécuter des logiciels sur n'importe quelle machine, indépendamment du système d'exploitation.

Je me suis intéressé au traitement de l'empreinte biométrique la plus courante, la photo d'identité. L'objectif était d'être capable de stocker une image de dimension fixée dans la carte et de la restituer au client puisque les applications cartes à puce fonctionnent selon le modèle client serveur. Le point central de l'étude réalisée repose sur la compréhension fine du dialogue à mettre en oeuvre entre d'une part, l'application client (écrite en pur Java) et la carte java intégrant l'applet téléchargée (écrite en Javacard). L'application client gère ainsi l'aspect graphique et fonctionnel du dialogue. On a défini une méthode de traitement par blocs élémentaires adaptée aux spécifications particulières de la carte on a réussi actuellement une entrée et une restitution d'une image binaire au format GIF. Les applications potentielles de cette première étude sont déjà nombreuses dans le domaine bancaire ou celui du contrôle d'accès car on peut associer à chaque transaction un affichage de la photo du porteur de la carte. L'écriture en Java présente également beaucoup d'attrait pour les évolutions futures !

# Chapitre 1: Généralités sur la carte à puce

# Introduction

Au cours de ce chapitre, on va décrire le domaine spécifique des cartes à puce. Pour des personnes non averties de ce sujet, on verra brièvement la technologie et l'histoire des cartes à puce depuis les dispositifs traditionnels consacrés aux applications spécifiques jusqu'aux plates-formes ouvertes permettant le téléchargement de nouveaux services tout le long de la vie de la carte. La première section devrait convaincre les lecteurs de la nature particulière des cartes à puce. La deuxième section présente l'état de l'art des systèmes à cartes à puce courants.

## 1.1 Le domaine spécifique des cartes à puce

Les cartes à puce forment un domaine spécifique qui présente plusieurs caractéristiques particulières à plusieurs niveaux : constitution interne, interfaces, cycle de vie et applications.

### 1.1.1 L'intérieur de la carte à puce

Une carte à puce est un rectangle de plastique, ayant la taille d'une carte de crédit, dans laquelle un simple microcontrôleur est intégré. Habituellement, les microcontrôleurs des cartes contiennent un microprocesseur (les 8-bit sont les plus répandus, mais les processeurs 16-bit et 32-bit peuvent maintenant être inclus) et différents types de mémoires: RAM (pour des données d'exécution), ROM (dans laquelle le système d'exploitation et les applications de base sont stockés), et EEPROM (dans laquelle les données persistantes sont stockées). Puisqu'il y a des contraintes fortes de taille, les quantités de mémoire sont relativement réduites. La plupart des cartes à puce vendues aujourd'hui contiennent au plus 512 octets de RAM, 32 Koctets de ROM, et de 16 Koctets d'EEPROM. Habituellement, la carte contient également quelques capteurs (de chaleur, de tension, etc.), qui sont utilisées pour mettre la carte hors tension quand elle est d'une façon ou d'une autre physiquement attaquée.

.

### 1.1.2 Les interfaces de carte

Afin d'être utilisable, une carte à puce doit être insérée dans un lecteur de cartes, qui fournit l'alimentation et le signal d'horloge. Le lecteur de cartes est généralement couplé à un dispositif d'affichage appelé terminal. Le terminal exécute un programme client qui peut accéder à la

carte à puce. N'importe quelle communication entre le terminal et la carte passe par le lecteur de cartes (Figure1) sous forme de messages échangés du terminal à la carte (commandes), et respectivement, de la carte au terminal (réponses). Tous ces aspects de base sont fortement normalisés, puisque des cartes sont censées être utilisables avec un large éventail de dispositifs. La famille des normes ISO 7816 est la référence. Elles normalisent les dispositifs, les positions des contacts sur la carte jusqu'au protocole de transport qui est employé pour communiquer entre la carte et le lecteur, comme le format des messages échangés entre le terminal et la carte.

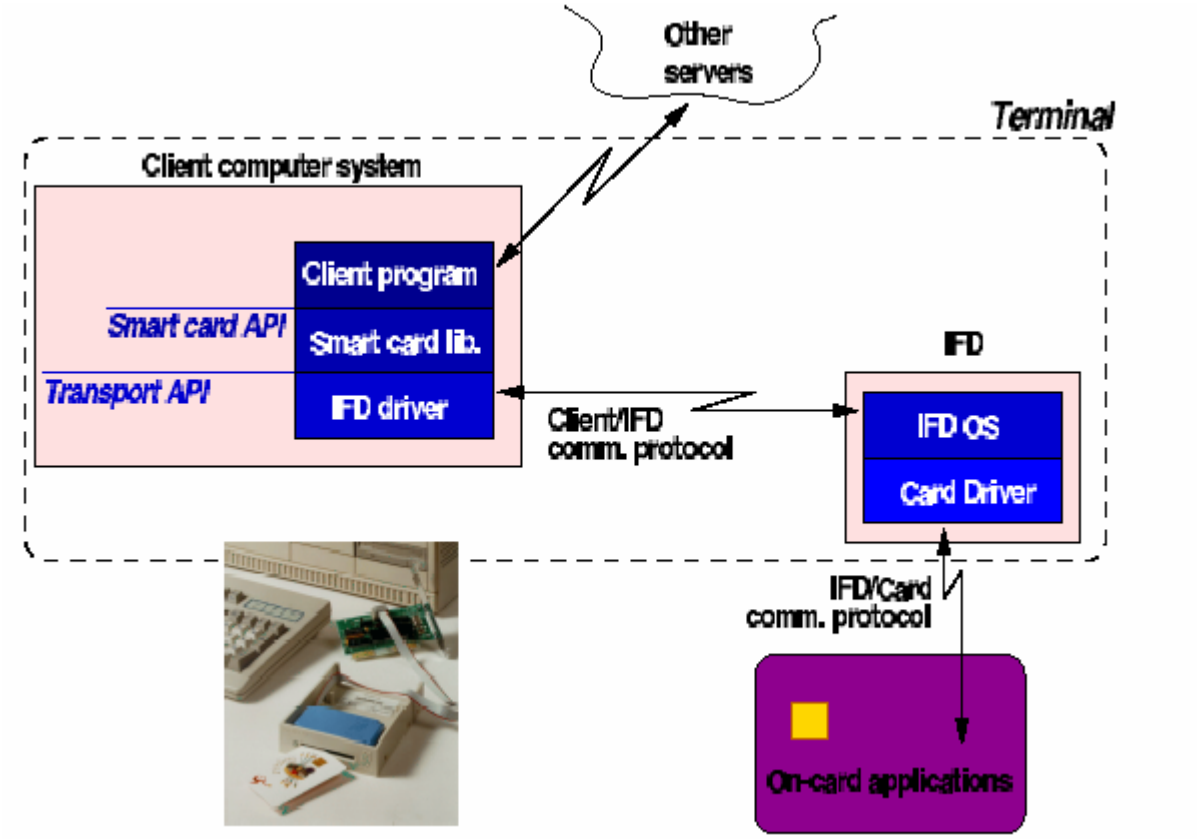



**Figure 1: Interface de la carte à puce**

### 1.1.3 Cycle de vie de la carte à puce

Une application carte est généralement déployée par un émetteur d'application qui obtient des cartes des fournisseurs. Les fournisseurs sont responsables de l'écriture du logiciel de base permanent dans la carte. Cette première étape du cycle de vie de la carte s'appelle la conception du masque. Elle consiste à inscrire dans la ROM le système d'exploitation de base qui

effectuera des commandes entrantes, et, pour les cartes à puce ouvertes (voir la section 1.2.2), la machine virtuelle et les bibliothèques d'api. La prochaine étape, appelée l'initialisation, se rapporte à charger toutes les données communes à l'application, par exemple, l'émetteur et les noms de l'application, dans la mémoire non volatile de la carte (généralement, l'EEPROM). Quand la carte est assignée à une personne, le processus de personnalisation ajoute au contenu d'EEPROM les données personnelles du propriétaire de la carte, par exemple, son nom et son code PIN. A ce stade, la carte à puce est devenue opérationnelle. Elle peut être insérée dans un lecteur de cartes et puis recevoir des commandes d'une application fonctionnant dans un terminal. Pour les cartes à puce ouvertes, l'application du terminal peut également télécharger de nouveaux programmes ou données sur la carte.

### 1.1.4 Les applications de la carte à puce

Une carte à puce peut être regardée comme " coffre-fort de données``, puisqu'elle stocke des données d'une façon sécurisée et elle est employée pendant des transactions courtes. Son circuit électronique est la base de sa sûreté.

Le fait que le circuit dans une carte est situé sous les contacts et que tous les composants soient sur le même circuit rend une attaque physique tout à fait difficile. Le logiciel est la deuxième barrière pour sa sûreté. Les programmes du circuit ne sont habituellement conçus ni pour retourner ni pour modifier l'information sensible avant d'être sûre que l'opération est autorisée. En fait, la plupart des applications utilisent les cartes à puce pour stocker des données sans risque, ou pour traiter des données sensibles. La plupart des cartes à puce incluent un certain soutien de fonctions cryptographiques, qui leur permet de sécuriser leurs transactions avec le monde extérieur. Plus pratiquement, des cartes sont souvent employées pour stocker un certain genre de devise (argent ou unité), pour enregistrer l'information très personnelle (comme des données médicales), ou pour identifier une personne. Avec les cartes à puce ouvertes, la fonctionnalité de la carte peut être prolongée par le téléchargement de nouveaux programmes ou de nouvelles données sur la carte.

## 1.2 Situation actuelle des cartes à puce

### 1.2.1 Systèmes à cartes à puce Traditionnels

Le domaine spécifique des cartes à puce est proche du domaine des circuits intégrés. Comme ces dispositifs, les cartes à puce sont visées par le marché de l'électronique grand public, qui exige toujours plus de performance et de flexibilité. Les méthodes, les langages et les outils pour développer un système à cartes à puce partagent certaines caractéristiques avec ceux du domaine de l'électronique. Jusque récemment, des codes de carte à puce ont été écrits en assembleur codé. Tous les programmes (pilotes, système d'exploitation, bibliothèques, applications) ont été développés dans un circuit monolithique de code brûlé dans la ROM de la carte à puce. Par conséquent, non seulement les systèmes à cartes traditionnels sont difficiles de développer (le langage de programmation de bas niveau, microcontrôleur de dispositif très réduit, le code spécifique pour chaque microprocesseur) mais également ils ne peuvent pas soutenir l'évolution de leurs applications puisque tout le code d'application est inscrit pour toujours dans la ROM. D'ailleurs, la production d'un programme consacré à un matériel spécifique et avec des fonctions ad hoc pour le domaine d'application consomme la majeure partie du cycle de développement pour la carte. Afin de publier une application de carte, il est nécessaire (i) d'écrire les caractéristiques précises, (ii) d'écrire ou réécrire le logiciel de base (apparenté à un système d'exploitation) pour les plateformes probablement multiples, (iii) de développer des fonctions spécifiques pour l'application, et (iv) de vérifier tout d'abord ce logiciel avant de le déployer sur des milliers ou des millions de cartes. Ce processus est long et coûteux. Pendant cette longue période, elle limite également sévèrement la capacité d'un émetteur de carte de déployer rapidement de nouvelles applications selon les besoins du marché.

### 1.2.2 Nouveaux Systèmes à cartes à puce Ouverts

Afin de faire face aux besoins du marché (pour simplifier, flexibilité pour les applications de la carte) des nouvelles générations de cartes à puce (appelées les cartes à puce ouvertes) ont émergé pendant les deux dernières années. La plupart des efforts notables vers de tels systèmes à cartes à puce sont la carte de Java, le MultOS, et la carte à puce pour Windows qui offrent

aux créateurs d'application une occasion pour créer des applications sur une base commune de code. Ils contiennent une plate-forme pour (c-à-d., sur demande) le stockage dynamique et l'exécution du contenu exécutable téléchargé, qui est basé sur une machine virtuelle pour la portabilité à travers les microcontrôleurs intelligents multiples de carte et pour les raisons de sécurité (Figure2).

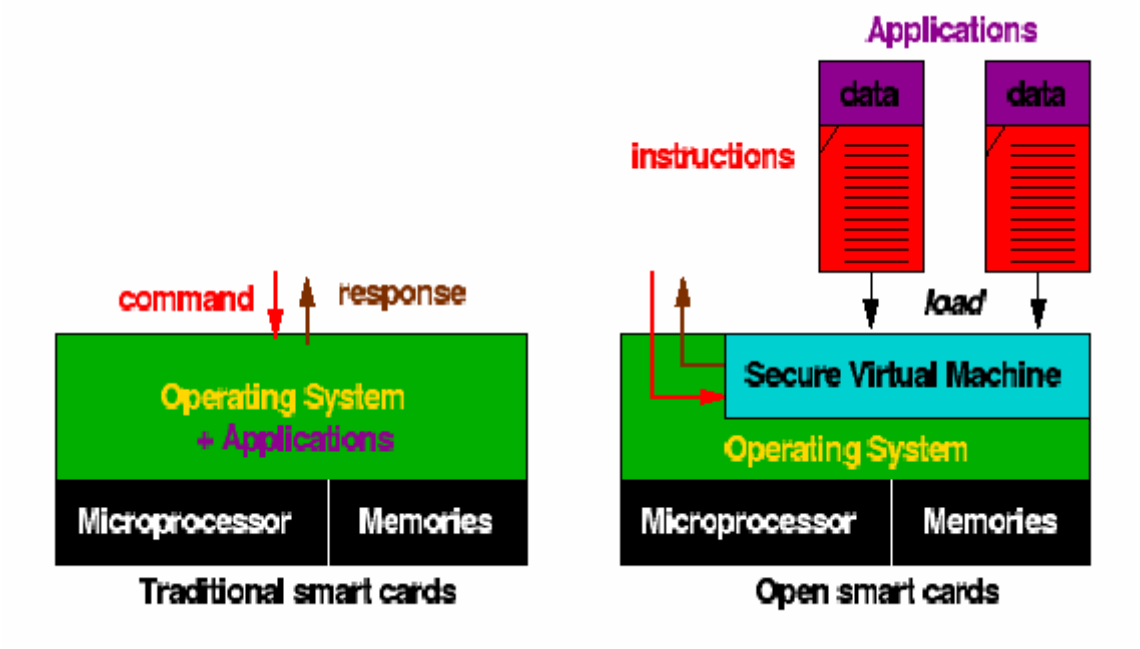



**Figure 2: De la carte à puce traditionnelle à la carte ouverte**

Tandis que ces nouvelles cartes à puce apportent des solutions concernant les besoins du marché, elles présentent également de nouveaux problèmes pour les fabricants de cartes à puce. Elles fournissent des solutions pour des créateurs d'application de carte en leur permettant de programmer dans des langages de haut niveau, sur une base commune du logiciel (une machine abstraite et des interfaces de programme d'application) qui isole leur code des bibliothèques de matériel spécifique et de système d'exploitation. Dans ce sens, elles peuvent réduire rigoureusement le temps d'obtenir de nouvelles applications. Elles tendent également à soutenir la flexibilité et l'évolution des applications en permettant le téléchargement du contenu exécutable dans des cartes à puce déjà déployées. Cette caractéristique exige plus de sophistication dans les techniques de sécurité appliquées aux programmes (en particulier,

l'attention doit être apportée à la malveillance qui pourrait potentiellement endommager le système entier de la carte).

S'assurer qu'un programme ne peut pas endommager un système consiste en lui refusant l'accès à d'autres zones de mémoire que ceux réservés pour son exécution et pour son code et données (retenue). La retenue se fonde sur des contrôles d'accès aux zones de mémoire. Il vaudrait mieux associer la retenue à un mécanisme de protection qui vérifie également que chaque instruction accède à des données en respectant leurs types. En fait, une bonne protection consiste à vérifier que chaque instruction qui accède à une zone de mémoire est conforme avec la définition des données stockées dans ce secteur (c-à-d., son type, ou sa classe dans un système orienté objet). Cette technique postérieure a été popularisée dans la langue de Java avec l'étape de vérification exécutée par la machine virtuelle chaque fois qu'elle charge une nouvelle classe d'une source non sécurisée.

Dans le domaine des cartes à puce ouvertes, la sécurité est généralement assurée par l'utilisation de trois techniques: pour la carte de Java, (1) un convertisseur de vérification outre de la carte effectue une vérification statique de la sûreté du type, ou (2) une machine virtuelle défensive effectue la vérification au temps d'exécution; pour d'autres, (3) aucune vérification de type n'est effectué et la confiance est seulement basée sur la retenue des applications. Pour les cartes à puce traditionnelles (tout le code étant brûlé dans leur ROM), ces techniques ne sont pas employées parce que la confiance en programmes est basée sur le fait qu'ils ont été examinés et vérifiés avant la livraison des cartes.

## 1.3 Conclusion

L'approche traditionnelle pour programmer les cartes à puce ne permet pas la création du code exécutable stockable et exige des programmeurs ayant une expérience dans les langages de programmation bas niveau. Ceci ne permet pas aux fabricants de carte de répondre rapidement aux changements du marché et limite la flexibilité des applications sur les cartes à puce. La programmation de cartes à puce ouvertes fournit une approche plus dynamique aux applications de carte. Les langages haut niveau et les mécanismes de sécurité servent de base à la programmation des cartes à puce ouvertes. La carte de Java fournit à ces deux dispositifs son langage orienté objet et son environnement d'exécution sécurisé.

# Chapitre 2: La technologie de la carte de Java

# Introduction

Dans ce chapitre, nous passons en revue la plateforme émergente et prometteuse de développement de carte à puce ouverte fournie par les caractéristiques de la carte de Java. Nous donnons une vue d'ensemble des processus impliqués dans la création d'une application basée sur la carte de Java.

Nous décrivons les limites des caractéristiques de la carte de Java pour fournir un cadre de programmation efficace et commode pour l'interaction entre les applets de carte de Java et le monde extérieur. Puis, nous montrons comment ces limites nous permettent de définir une méthodologie de développement basée sur les principes de l'orienté objet distribué appliqués à la carte de Java.

## 2.1 Vue d'ensemble de la carte Java

Pendant les vingt dernières années, les cartes à puce ont évolué depuis les dispositifs dédiés simples aux plateformes de calcul ouvertes. La recherche se concentrait à fournir les cartes à puce programmables qui permettraient à des programmeurs d'écrire leurs applications carte sur une machine virtuelle exécutant le code dans un environnement portatif et sécurisé. Avec l'arrivée de la carte de Java, favorisé par Sun Micro system et les forums de carte de Java, la technologie de carte à puce a été rendue accessible à un grand nombre de programmeurs. Vu que la plupart de ces programmeurs sont nouveaux à programmer sur la carte à puce, ils peuvent se servir de leur connaissance dans Java pour aborder les problèmes spécifiques de la programmation de carte à puce.

Les caractéristiques de la carte de Java 2.0 fournissent l'interopérabilité du code source écrit pour la carte de Java en définissant le support d'exécution (la machine virtuelle de carte de Java), le langage de Java, et les APIs disponibles pour les programmeurs. Cette interopérabilité est basée (comme sur toute plateforme de Java) sur la combinaison de l'indépendance d'unité centrale de traitement (par l'utilisation de la machine virtuelle de carte de Java), et l'indépendance du système d'exploitation (par l'utilisation d'une API standard de la carte de java). Concernant la sécurité de la coexistence de plusieurs applet sur la même carte, elle est assurée en employant le principe du firewall qui isole des applets de carte l'une de l'autre. On affirme que c'est de la responsabilité de l'implémentation de la machine virtuelle de la carte de

Java de s'assurer qu'aucun applet ne peut utiliser, accéder, ou ne modifier le contenu d'un objet possédé par un autre applet.

Les caractéristiques de la carte de Java 2.1, récemment rendue disponible, dégagent les spécifications des API de la carte de Java dans une manière dont maintient la fonctionnalité de base de 2.0 mais clarifie et améliore quelques points. En outre, la carte de Java 2.1 ajoute deux nouvelles caractéristiques qui améliorent la portabilité des applications parmi différentes implémentations de la carte de Java. La première spécification est le JCRE (Java Card 2.1 Runtime Environment ) qui indique les détails de l'environnement d'exécution de la carte de Java et permet aux programmeurs de savoir que leurs programmes s'exécuteront identiquement d'une plateforme de carte de Java à une autre. La deuxième spécification est la machine virtuelle de la carte de Java 2.1 qui définit les normes exigées pour assurer la portabilité binaire des programmes de la carte de Java. Elle inclut le sous-ensemble soutenu du langage Java (présente déjà dans 2.0), l'ensemble d'instruction de la machine virtuelle de la carte de Java (nouvelle dans 2.1), et les formats de dossier ou de fichier (la représentation binaire des programmes) utilisés pour installer des programmes dans les implémentations de la carte de Java.

Le rapport est à propos de la programmation de la carte de Java. Ainsi, nous sommes principalement concernés par les caractéristiques d'Api qui ne changent pas considérablement de 2.0 à 2.1. Les changements seront indiqués dans le rapport mais ne vont pas influencer le contenu du rapport et les exemples fournis après. Cependant, les lecteurs doivent se rappeler les différences entre 2.0 et 2.1 le moment où ils ont l'intention de déployer des applications avec différents dispositifs qui mettent en application la technologie de la carte de Java. Avec les produits 2.0, la portabilité de leurs programmes de carte de Java est limitée à leur source. En contre partie, les produits 2.1 devraient soutenir la portabilité binaire de leurs programmes.

## 2.2 La plateforme de la carte de Java

### 2.2.1 L'environnement d'exécution de la carte de Java (JCRE)

Une carte de Java est simplement définie comme carte à puce qui est capable d'exécuter des programmes de Java, appelé des applets de carte ou avec plus de concision des applets. Elle est basée sur un environnement d'exécution de carte de Java (JCRE) qui inclut l'implémentation de

la machine virtuelle de la carte (JCVM), les classes API de la carte de Java, et des services de support. Plus spécifiquement, les éléments principaux du JCRE (Figure3) sont:

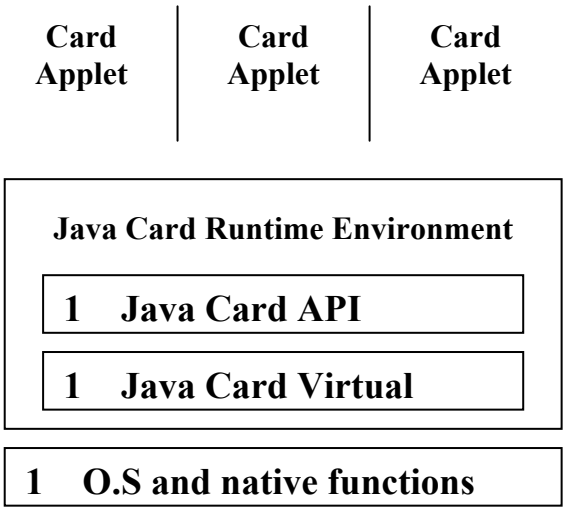



**Figure 3:L'architecture de la carte Java**

**a- La machine virtuelle de la Carte de java**

Une couche qui cache les composants de bas niveau (langage machine, système d'exploitation et fonctions indigènes) avec un interprète d'un sous-ensemble de bytecodes de Java. Elle ne soutient pas les mécanismes suivants: le chargement dynamique de classe, la direction de la sécurité, les threads et la synchronisation, le clonage d'objet, la collection d'ordures et l'achèvement, les types char, long, float, double, et les tableaux avec plus d'une dimension.

**b- L'API de la carte de java**

Une interface commune de programmation définie par quelques paquets (packages). Le premier définit un sous-ensemble strict du paquet standard de java.lang, reflétant les restrictions de langue énumérées ci-dessus. En fait, il y a seulement les classes object (avec seulement le constructeur et la méthode equals), Throwable et quelques classes d'exception (avec seulement le constructeur de défaut sans un paramètre String). Les autres paquets fournis sont directement liés aux normes de la carte à puce ou liés à la cryptographie:

- javacard.framework et javacardx.framework définient respectivement les classes pour la manipulation des dispositifs spécifiques de la carte à puce tels que le format des messages échangés avec le terminal et une norme de la carte pour la gestion des fichiers.

- javacardx.crypto et javacardx.cryptoEnc soutenants les fonctions cryptographiques.

Les principaux changements de caractéristiques d'API de la carte de Java 2.1 sont la suppression du javacardx.framework, et la nouvelle structure des paquets cryptographiques.

**c- Les services de support**

Ils sont responsables des paramètres suivants de la carte de Java:

- La durée de vie du JCVM. Une fois que la carte est initialisée, le JCVM ne s'arrête jamais complètement, jusqu'à ce que la carte soit jetée. Ainsi, les objets alloués par le JCVM demeurent disponibles durant toute la vie de la carte. En fait, ces objets sont stockés constamment dans EEPROM, et ce dispositif est employé pour expliquer cette persistance de la façon la plus simple possible.
- La durée de vie des applets. Les applet doivent être chargés, installés, choisis. . . L'état de l'applet (Figure2.2) est contrôlé par le JCRE par l'intermédiaire des méthodes publiques de l'applet install, select, deselect, et process.
- Des objets passagers sont créés par l'intermédiaire de facteurs spéciaux puisque le mot-clé transient de Java n'est pas soutenu par la carte de Java.
- Le choix de l'applet actif qui exécutera et traitera les commandes venant du terminal.
- L'isolement d'applet (par un firewall) et le blocage d'objets partagés entre les applets sur le firewall.
- Le soutien des transactions atomiques, de sorte que des données de carte soient reconstituées à leur état original de pré-transaction si la transaction n'accomplit pas normalement.
- Le procédé d'installation d'applet

La figure2.2 suivante récapitule les différents états d'applet tout le long de sa vie dans la carte. La transition d'un état à l'autre est strictement commandée par le JCRE.

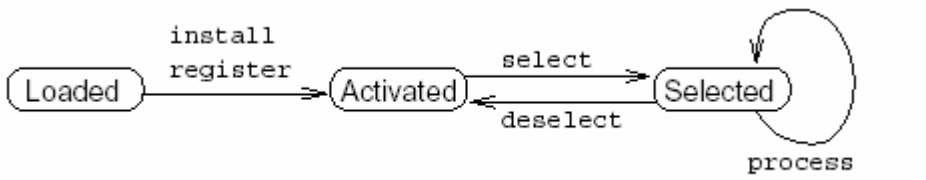



**Figure 4:Diagramme d'état d'applet de la carte de Java**

Les caractéristiques de la carte de Java décrivent seulement les dispositifs de noyau du JCRE, c.-à-d., ceux qui sont exigés afin de réaliser le niveau désiré de l'interopérabilité qui est la programmation des cartes à puce basées sur Java. De telles caractéristiques ne sont pas prévues pour indiquer complètement le comportement d'un système à cartes à puce. Maintenant, les constructeurs de carte devraient indiquer un environnement plus complet, comprenant des mécanismes tel que le chargement sécurisé d'applet, et cycles de vie complets de carte et d'applet.

### 2.2.2 La Machine Virtuelle De La carte De Java

La machine virtuelle de la carte de Java (JCVM) diffère d'une machine virtuelle typique de Java (JVM) par les dispositifs suivants :

**a- La gestion de la mémoire**

Le JCVM est une machine persistante. Dans une carte de Java, tous les objets sont, par défaut, persistant; ils sont stockés dans la mémoire non volatile de la carte. En outre, les classes sont chargées et initialisées seulement une fois dans le JCVM, où elles demeurent actives jusqu'à être rejetées. En conclusion, le mécanisme de collection d'ordures n'est pas soutenu, rendant impossible la reprise de l'espace alloué à un objet.

**b- Vérificateur de bytecode**

L'implémentation typique d'un JVM est faite d'un vérificateur de bytecode, d'un chargeur de classe, et d'un interprète de bytecode. Le vérificateur vérifie qu'un dossier de classe est un dossier valide de classe de Java. Le chargeur charge des classes dans le système. Et, l'interprète exécute l'application. Un vérificateur de bytecode est un morceau de logiciel très complexe qui ne peut pas s'adapter sur une carte à puce. Ainsi, le vérificateur de Java est utilisé et exécuté sur une station de centre serveur sur les dossiers typiques de classe de Java contenant le bytecode d'applet de la carte de Java.

**c- Convertisseur de bytecode**

Après la vérification des dossiers de classe en dehors de la carte, ces dossiers sont transformés en une forme plus appropriée aux cartes à puce. Un convertisseur est employé pour transformer un ensemble de dossiers de classe en un dossier chargeable qui contient toutes les classes d'un paquet. La transformation inclut le nom de la résolution, l'enchaînement, aussi bien que quelques optimisations de bytecode.

### d- Chargeur de bytecode

Comme le chargement du bytecode peut seulement être effectué après es étapes de la vérification et de la conversion, le chargeur de JCVM est dédoublée en deux parts: un off-card qui envoie le dossier chargeable dans la carte, et une partie on-card qui installe les classes sur la mémoire de la carte.

En conclusion, l'image claire d'un JCVM est qu'elle est un JVM mise en application en tant que deux morceaux séparés (Figure2.3). Le premier morceau du JVM exécute l'off-card sur un PC ou un poste de travail. Il contient les pièces d'off-card de JCVM qui sont le vérificateur de bytecode, le convertisseur de bytecode, et le chargeur d'off-card. Le deuxième morceau du JCVM exécute l'on-card. Il inclut le chargeur d'on-card et l'interprète de bytecode. Afin d'assurer la continuité du JCVM (le fait que les pièces d'on-card peuvent faire confiance à l'off-card exécuté par tâches, principalement le procédé de vérification), une signature cryptographique peut être associée au dossier chargeable, qui montre que les classes ont passé correctement par les outils d'off-card.

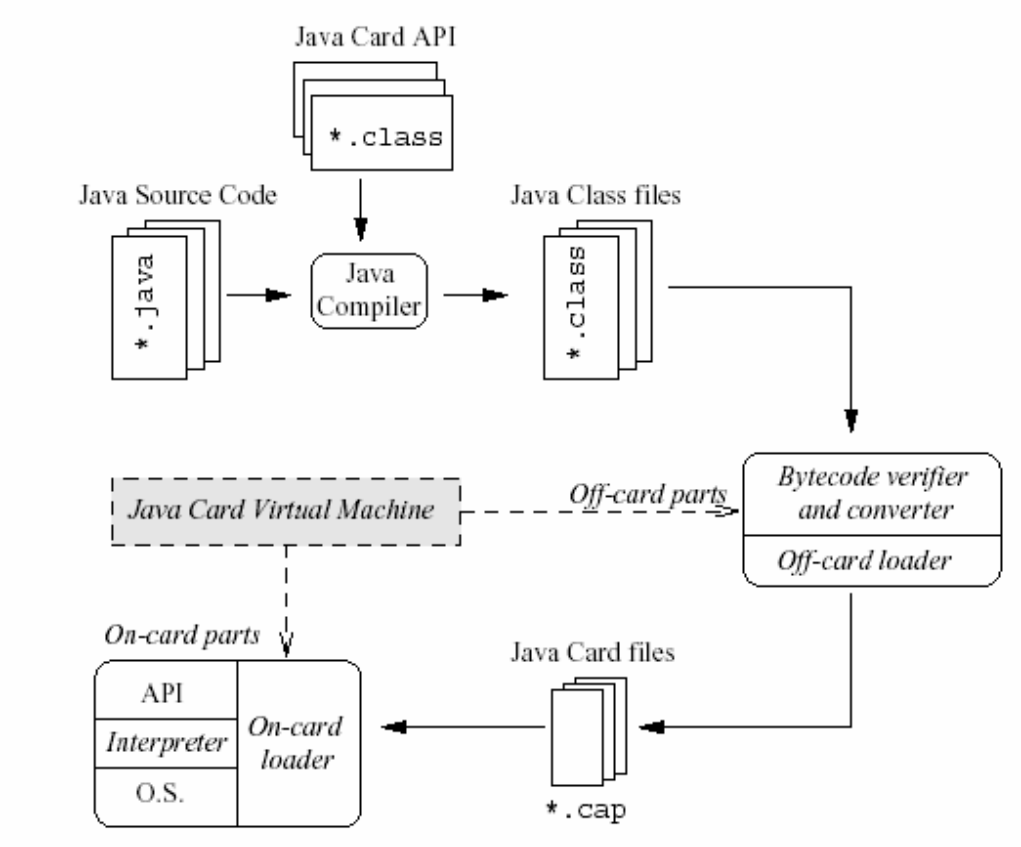



**Figure 5: Structure de la machine virtuelle de la carte java**

## 2.3 Au-delà de la carte java

Les spécifications de la carte java décrivent une version limitée de la plate-forme et du langage java, dédiée aux cartes à puce. Définir la communication entre l'application de la carte et le monde extérieur comme échange de messages APDU force les programmeurs à désigner un protocole convenable pour chaque application qu'ils écrivent.

L'application client et l'applet de la carte doivent implémenter un code signifiant permettant de chiffrer et de déchiffrer l'information échangée par l'intermédiaire du protocole définit.

Cette approche est gênante et difficile à programmer, et elle force les programmeurs à se concentrer sur la définition du protocole bas niveau dédié à la carte à puce plutôt que sur la conception de leur application.

A propos de l'implémentation de la carte java (appelée GemXpresso), il a été proposé une méthodologie de développement qui permet aux programmeurs de la carte et des applications client de se libérer ou se débarrasser de la définition du protocole de communication entre la carte et le terminal. Cette méthodologie est basée sur la représentation de l'architecture client-serveur établie par le terminal et la carte comme système d'objet distribué. La carte java est considérée comme un serveur exécutant une application décrite par une interface.

L'applet de la carte est une classe qui implémente cette interface. Le programme client communique avec l'applet de la carte selon le désir de l'interface. L'interface est une sorte de contrat qui lie un srveur à son client. Le serveur garanti qu'il répondra, comme attendu, aux méthodes définies dans l'interface.

Pour la communication entre l'application client et l'applet de la carte, on a défini un protocole dit DMI (Direct Method Invocation), qui résume les requêtes client aux applets de la carte. Le protocole DMI lie la carte au terminal et définie la manière avec laquelle l'applet de la carte te l'application client communiquent. Le protocole est manipulé par le système d'exploitation de la carte java, rendant le code source de l'applet de la carte entièrement indépendant de lui. Du côté client le protocole est manipulé par un programme de proxy qui est automatiquement généré par l'interface. Le proxy représente l'objet écarté (l'applet de la carte) du côté client en offrant à l'application client l'interface implémentée par l'applet. Le proxy implémente l'interface mais il contient le code nécessaire pour invoquer les méthodes de l'applet de la carte utilisant le protocole DMI.

## 2.4 Conclusion

Les spécifications de la carte java définissent un environnement de programmation, basé sur la plateforme et le langage java, qui peut être utilisé pour programmer les cartes à puce d'une manière inter opérable. La version ou la technologie 2.0 est centrée autour de la définition d'un API, et elle inclut de même la description d'un environnement d'exécution pour la carte et d'une machine virtuelle de la carte java, dans le but de garantir l'interopérabilité des programmes de la carte java. Néanmoins, cette spécification laisse les programmeurs s'occuper Des messages de communication bas niveau dans leurs programmes. Pour l'implémentation de la carte, il a été proposé une approche qui a en fait la même esprit que la squelette RMI(Remote Method Invocation) de java, ou plus précisément que le mécanisme RPC(Remote Procedure Call). L'application client utilise un proxy (local object) afin de pouvoir accéder et utiliser l'applet de la carte ( remote object) tout en invoquant des méthodes définies dans l'interface de l'applet. Un protocole spécifique (le protocole DMI) entre le terminal et la carte est utilisé pour cette raison. Le protocole DMI est manipulé par le système d'opération de la carte du coté serveur et par un proxy du coté client. Le but de al squelette carte java/DMI est de simplifier la tâche d'un programmeur qui veut concevoir et écrire une nouvelle application qui utilise la carte java. Considérant la carte comme un composant d'un système d'objet réparti délivre les programmeurs de se charger de l'émission du protocole, et leur permet de passer plus de temps au niveau de la performance et la fonctionnalité de l'application.

# Chapitre3: Développement d'un applet pour la carte de Java

# Introduction

La technologie de la carte de Java fournit la plus petite plateforme de Java pour les dispositifs à mémoire limitée comme les cartes à puce. Le contenu de ce chapitre représente un guide qui permet d'apprendre étape par étape les concepts de programmation que les programmeurs d'Apis devraient employer en développant des applets de la carte de Java, tout en illustrant ces concepts à travers un exemple d'applet écrit en Java.

Ce chapitre est prévu pour les programmeurs de Java qui souhaitent prolonger leurs efforts de développement sur la plateforme de la carte de Java. Il fournit également une excellente introduction pour des programmeurs ayant une expérience dans le développement d'application pour la carte à puce, mais accoutumé à la programmation en C ou en assembleur.

## 3.1 Architecture de l'applet

Comme avec n'importe quel développement d'application software, avant de s'asseoir et écrire un applet de la carte de Java, on doit tout d'abord passer par une phase de conception. Au cours de cette phase, on définit l'architecture de l'applet. Quatre étapes comportent la phase de conception de l'applet:

1. Indiquer les fonctions de l'applet.
2. Demander et attribuer des AIDs bien pour l'applet que pour le paquetage contenant la classe de l'applet.
3. Désigner la structure de classe des programmes d'applet.
4. définir l'interface entre l'applet et l'application du terminal.

Dans les sections suivantes, on emploiera l'exemple d'un applet wallet pour avoir une idée détaillée sur chacune des étapes du processus de conception d'applet.

## 3.2 Indication des fonctions de l'applet

L'exemple de l'applet wallet stockera l'argent électronique et soutiendra les fonctions de crédit, de débit, et de vérification du montant disponible.

Pour aider à empêcher l'utilisation non autorisée de la carte, cette dernière contient un algorithme de sécurité. Cet algorithme exige de l'utilisateur d'entrer dans un code PIN, une chaîne de caractère de huit chiffres au plus. L'utilisateur de la carte tape son code PIN sur un bloc notes (keypad) relié au lecteur CAD (card acceptance device). L'algorithme de sécurité cause la carte à se fermer ou à se bloquer après trois tentatives non réussies d'entrer le code PIN convenable. Le code PIN est initialisée selon les paramètres d'installation quand l'applet

est installé et créé.

Le code PIN doit être vérifié avant que n'importe quelle transaction de crédit ou de débit ne puisse être exécutée.

Pour la simplicité, disons que le montant maximum disponible dans la carte est de $32.767, et qu'aucune transaction de crédit ou de débit ne peut excéder $127. Ainsi, les variables de Java de type short et byte peuvent représenter le montant disponible du wallet et la somme de chaque transaction, respectivement.

* Un applet wallet réel exigerait un mécanisme de sécurité beaucoup plus sophistiqué de sécurité pour empêcher l'accès non autorisé dans la wallet.

## 3.3 Spécification des AIDs

La plupart des applications dont on est familiarisé sont appelées et identifiées par un nom sous forme d 'une chaîne de caractère. En technologie de carte de Java, cependant, chaque applet est identifié et sélectionné par un AID. En outre, pour chaque paquetage de Java est assigné un AID. Ceci est du au fait qu'un paquetage, une fois chargé sur une carte, est lié avec d'autres paquets, qui ont été déjà placés sur la carte par l'intermédiaire de leurs AIDs. Cette convention d'appellation conforme avec les spécifications de la carte à puce comme défini en ISO7816.

Un AID est une séquence d'octets entre 5 et 16 octets de longueur. Son format est dépeint dans le tableau suivant :

| **Application identifier (AID)** | |
|---|---|
| **National registered application provider (RID)** | **Proprietary application identifier extension (PIX)** |
| 5 bytes | 0 to 11 bytes |

**Figure 6:Les formats des AIDs**

Les classes de Java de l'applet wallet sont définies dans un paquetage Java. Les AIDs factices de l'applet wallet et du paquetage d'applet sont définies comme illustré dans le tableau suivant :

| Package AID | | |
|---|---|---|
| **Field** | **Value** | **Length** |
| RID | 0xF2, 0x34, 0x12, 0x34, 0x56 | 5 bytes |
| PIX | 0x10, 0x00, 0x00 | 3 bytes |
| **Applet AID** | | |
| **Field** | **Value** | **Length** |
| RID | 0xF2, 0x34, 0x12, 0x34, 0x56 | 5 bytes |
| PIX | 0x10, 0x00, 0x01 | 3 bytes |

**Figure 7 :Les AIDs de l'applet wallet et du paquetage de l'applet**

L'AID du paquet et l'AID de l'applet ont la même la valeur du RID; leurs valeurs de PIX diffèrent au niveau du dernier bit.

## 3.4 Définition de la structure des classes et des méthodes de l'applet

Une classe d'applet de la carte de Java doit s'étendre de la classe javacard.framework.Applet. Cette classe est la superclasse pour tous les applets résidants dans une carte de Java. Elle définit les méthodes communes qu'un applet doit soutenir afin d'interagir avec le JCRE pendant sa durée de vie.

Le tableau suivant énumère les méthodes publiques et protégées définies dans la classe javacard.framework.Applet:

| Method summary | |
|---|---|
| `public void` | `deselect ()`<br><br>appelée par le JCRE pour informer l'applet courante sélectionnée qu'un autre (ou la même) applet sera sélectionner . |

| | |
|---|---|
| public static void | install (byte[] bArray, short bOffset, byte bLength)<br><br>Le JCRE appelle cette méthode statique pour créer une instance de<br>L' Applet subclass. |
| public abstract void | process (APDU apdu)<br><br>c'est une méthode principale qui reçoit et transfère tous les APDUs de l'applet tant qu'elle est sélectionnée. |
| protected final void | register ()<br>cette méthode est utilisée par l'applet pour enregistrer cette applet instance dans le JCRE et pour assigner l'AID de défaut dans le fichier du CAD à l'applet instance. |
| public boolean | select ()<br><br>appelée par le JCRE pour informer cette applet qu'elle a été sélectionnée. |

**Figure 8:Les méthodes Publiques et protégées de la classe javacard.framework.Applet**

La classe javacard.framework.Applet fournit un cadre pour l'exécution d'applet. Des méthodes définies dans cette classe sont appelées par le JCRE quand ce dernier reçoit des commandes APDU du lecteur CAD.

Après que le code d'applet ait été correctement chargé sur une carte de Java et lié avec d'autres paquets dans la carte, la vie d'un applet commence quand un applet instance est créé et inscrit à la table d'enregistrement du JCRE. Un applet doit mettre en application la méthode statique install() pour créer un applet instance et pour enregistrer l'instance dans le JCRE en appelant l'une des deux méthodes register().

La méthode install() prend un array de type byte comme paramètre. Cette array contient les paramètres d'installation pour initialiser ou personnaliser l'applet instance.

Un applet sur une carte de Java est à une étape inactive jusqu'à ce qu'il soit explicitement choisi. Quand le JCRE reçoit une commande SELECT APDU, il recherche dans sa table interne l'applet dont l'AID assortit celui indiqué dans la commande. Si une correspondance

est trouvée, le JCRE prépare la nouvelle applet pour être sélectionnée. Ce procédé de préparation se compose de deux étapes: D'abord, si une applet courante déjà sélectionnée est présente, le JCRE la désélectionne en appelant la méthode deselect(). L'applet effectue n'importe quel travail de nettoyage ou de comptabilité durant la méthode de deselect() avant qu'il entre dans l'étape inactive. Alors le JCRE appelle la méthode select () pour informer le nouvel applet qu'il a été choisi. Le nouvel applet exécute n'importe quelle initialisation nécessaire avant qu'il devienne réellement choisi. L'applet renvoie son accord à la méthode select () s'il est maintenant prêt à devenir actif et à traiter des commandes APDU. Autrement, l'applet retourne faux pour refuser sa participation, et si oui, aucun applet ne sera choisi. La classe javacard.framework.Applet fournit une implémentation par défaut des méthodes de select () et de deselect().

Une fois qu'un applet est choisi, le JCRE expédie toutes les commandes APDU (y compris la commande SELECT) à la méthode process () de l'applet. Au cours de la méthode process (), l'applet interprète chaque commande APDU et accomplit la tâche indiquée par la commande. Pour chaque commande APDU, l'applet répond au CAD en renvoyant une réponse APDU, qui informe le CAD du résultat de traitement de la commande APDU. La méthode process () de la classe javacard.framework.Applet est une méthode abstraite. Ce dialogue de commande et réponse continue jusqu'à ce qu'un nouvel applet soit choisi ou la carte soit enlevée du CAD. Une fois désélectionné, un applet devient inactif jusqu'à ce qu'il soit sélectionné une autre fois.

## 3.5 Interface entre un applet et son application terminale

Un applet fonctionnant dans une carte à puce communique avec l'application terminale dans le CAD en utilisant des APDUs (Application Protocol Data Units).

Essentiellement, l'interface entre un applet et son application terminale est un ensemble de commandes APDU qui sont convenues et soutenues par l'applet et l'application terminale.

### 3.5.1 Architecture

Pour communiquer avec une carte à puce, il faut écrire une application qui envoie des commandes à la carte à puce connectée au terminal à travers le lecteur de carte. Du terminal au lecteur, et du lecteur à la carte à puce, les données (commandes et réponses) sont transférées par l'intermédiaire de protocoles. Les protocoles de communication entre les cartes à puce et les lecteurs sont définis par la norme ISO comme T=0 (transmission de

caractère) ou T=1(transmission de blocs de caractères).

Le problème est qu'il n'y a pas de protocole standardisé entre le terminal et le lecteur.

Dans le but de permettre la description des commandes de la carte indépendamment des protocoles terminal-lecteur et lecteur-carte un format a été standardisé. Ce format définit le message de commande envoyé depuis l'application client, et le message de réponse retourné par la carte à l'application client sous forme d'APDU. Chaque fabricant fournit un driver pour transporter les messages APDU avec sa propriété terminal-lecteur protocole (Figure 9).

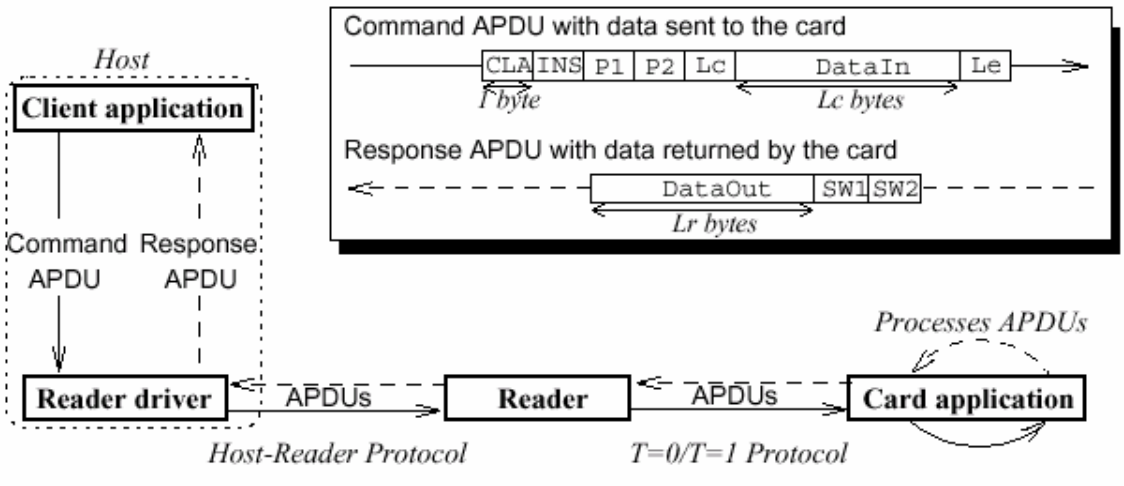



**Figure 9:Communication avec les APDUs**

## 3.5.2 Le protocole de communication (APDU)

Cette section fournit un sommaire des commandes APDU (les détails du protocole APDU sont spécifiés par la norme ISO 7816.) Les commandes APDU sont toujours des ensembles de paires. Chaque paire contient une commande APDU, qui indique une commande, et une réponse APDU, qui renvoie le résultat d'exécution de la commande. Dans le monde de carte, les cartes à puce sont des communicateurs réactifs -- c'est-à-dire, elles ne lancent jamais des communications, elles répondent seulement à des APDUs du monde extérieur. L'application terminale envoie une commande APDU par le CAD. Le JCRE reçoit la commande et sélectionne un nouvel applet ou bien, il passe la commande à l'applet actuellement sélectionné. L'applet actuellement sélectionné traite la commande et renvoie une réponse APDU à l'application terminale. Les commandes APDUs et les réponses APDUs sont échangées alternativement entre une carte et un CAD.

Le tableau suivant décrit des formats de commande et de réponse APDU.

**La commande APDU**

| en-tête obligatoire | | | | Corps optionnel | | |
|---|---|---|---|---|---|---|
| CLA | INS | P1 | P2 | Lc | Data field | Le |

- CLA (1 byte): la classe de l'instruction --- indique la structure et le format d'une catégorie de commandes et de réponses APDUs
- INS (1 byte): le code de l'instruction: spécifie l'instruction de la commande
- P1 (1 byte) and P2 (1 byte): les paramètres de l'instruction – fournit des qualifications pour l'instruction
- Lc (1 byte): Nombre d'octet présent dans le domaine de données de la commande
- Domaine de données (octet égale à la valeur de Lc): une séquence d'octet dans le domaine de données de la commande
- Le (1 byte): Maximum d'octet à accueillir dans le domaine de données de la réponse à la commande

**La réponse APDU**

| Corps optionnel | Trailer obligatoire | |
|---|---|---|
| Data field | SW1 | SW2 |

- Data field (variable length): une séquence d'octet reçue dans le domaine de données de la réponse
- SW1 (1 byte) et SW2 (1 byte): Status words – indique l'état de traitement dans la carte

**Figure 10: Les formats des Commandes et des réponses APDU**

### 3.5.3 Définition des commandes APDU

Un applet de carte de Java devrait soutenir un ensemble de commandes APDU, comportant une commande SELECT APDU et un ou plusieurs commandes APDUs de traitement.

- La commande SELECT demande au JCRE de sélectionner l'applet dans la carte.
- L'ensemble de commandes de processus définit les commandes que l'applet peut supporter ou soutenir. Celles-ci sont définies selon les fonctions de l'applet.

La technologie de carte de Java indique le codage de la commande SELECT APDU. Les réalisateurs d'applet sont libres de définir le codage de leurs commandes de processus. Cependant, les commandes de processus doivent être conformes à la structure décrite ci-dessus.

Structurellement, la commande SELECT et les commandes de processus sont des paires de la commande et de la réponse APDU.

Pour chaque commande APDU, l'applet devrait d'abord décoder la valeur de chaque champ dans la commande. Si les zones d'information facultatives sont incluses, l'applet devrait également déterminer leur format et leur structure. En utilisant ces définitions, l'applet sait interpréter chaque commande et lire les données. Il peut alors exécuter la tâche indiquée par la commande.

Pour chaque réponse APDU, l'applet devrait définir un ensemble de mots de statut (status words) pour indiquer le résultat de traitement de la paire de commande APDU. Pendant le traitement normal, l'applet renvoie le mot de statut de succès (0x9000, comme indiqué dans la norme ISO 7816). Si une erreur se produit, l'applet doit renvoyer un mot de statut autre que 0x9000 pour dénoter son état interne. Si la zone d'information facultative est incluse dans la réponse APDU, l'applet devrait définir quoi retourner.

Dans l'exemple d'applet wallet, l'applet soutient le crédit, le débit, et les fonctions de vérification du montant disponible. En outre, il doit soutenir la commande VERIFY pour la vérification du code PIN.

La commande SELECT et quatre commandes APDUs de processus pour l'applet wallet sont définies comme illustré dans le tableau suivant :

| **SELECT APDU command** | | | | | | |
|---|---|---|---|---|---|---|
| **Command APDU** | | | | | | |
| **CLA** | **INS** | **P1** | **P2** | **Lc** | **Data field** | **Le** |
| 0x0 | 0xA4 | 0x04 | 0x0 | 0x08 | 0xF2, 0x34, 0x12, 0x34, 0x56, 0x10, 0x0, 0x1 | N/A |
| la commande header (CLA, INS, P1, et P2) doit être codé comme dans le tableau ci-dessus, de cette façon le JCRE peut l'identifier comme une commande `SELECT` APDU. La zone de données contient l'AID de l'applet wallet. Le JCRE cherche dans sa table interne vis-à-vis de l'octet AID. Si une correspondance est trouvée, l'applet wallet est sélectionné. | | | | | | |

**Response APDU**

| Optional data | Status word | Meaning of status word |
|---|---|---|
| No data | 0x9000 | Successful processing |
| | 0x6999 | Applet selection failed: the applet could not be found or selected |

**VERIFY APDU command**

**Command APDU**

| CLA | INS | P1 | P2 | Lc | Data field | Le |
|---|---|---|---|---|---|---|
| 0xB0 | 0x20 | 0x0 | 0x0 | Length of the PIN data | PIN data | N/A |

- L'octet CLA dénote la structure de la commande
- L'octet INS (0x20) indique une instruction de vérification
- P1 et P2 ne sont pas utilisés, ils sont mis tous les deux à 0
- La zone de données contient la valeur du PIN

| Optional data | Status word | Meaning of status word |
|---|---|---|
| N/A | 0x9000 | Successful processing |
| | 0x6300 | Verification failed |

**CREDIT APDU command**

< code>**Command APDU**

| CLA | INS | P1 | P2 | Lc | Data field | Le |
|---|---|---|---|---|---|---|
| 0xB0 | 0x30 | 0x0 | 0x0 | 1 | Credit amount | N/A |

- La zone de données contient la somme de crédit.

< code>**Response APDU**

| Optional data | Status word | Meaning of status word |
|---|---|---|
| N/A | 0x9000 | Successful processing |

| | | |
|---|---|---|
| | 0x6301 | PIN verification required |
| | 0x6A83 | Invalid credit amount |
| | 0x6A84 | Exceed the maximum amount |

< code>**DEBIT APDU command**

< code>**Command APDU**

| CLA | INS | P1 | P2 | Lc | Data field | Le |
|---|---|---|---|---|---|---|
| 0xB0 | 0x40 | 0x0 | 0x0 | 1 | Debit amount | N/A |

- La zone de données contient la somme du débit.

**Response APDU**

| Optional data | Status word | Meaning of status word |
|---|---|---|
| N/A | 0x9000 | Successful processing |
| | 0x6301 | PIN verification required |
| | 0x6A83 | Invalid debit amount |
| | 0x6A85 | Negative balance |

< code>**GET BALANCE APDU command**

< code>**Command APDU**

| CLA | INS | P1 | P2 | Lc | Data field | Le |
|---|---|---|---|---|---|---|
| 0xB0 | 0x50 | 0x0 | 0x0 | N/A | N/A | 2 |

- The data field contains the balance amount

**Response APDU**

| Optional data | Status word | Meaning of status word |
|---|---|---|
| N/A | 0x9000 | Successful processing |

**Figure 11:Les commandes APDU de l'applet wallet**

En plus des mots de statut déclarés dans chaque réponse APDU, l'interface javacard.framework.ISO7816 définit un ensemble de mots de statut qui signalent des erreurs communes pour les applets.

### 3.5.4 Traitement de l'objet APDU

La classe javacard.framework.APDU résume les commandes APDU. Elle fournit une interface puissante et flexible pour permettre à des applets de manipuler des commandes APDU. La classe APDU est conçue pour cacher les complexités du protocole, ainsi les réalisateurs d'applet peuvent se concentrer sur les détails de l'application.
Quand le JCRE reçoit une commande APDU, il encapsule la commande comme un objet APDU et passe l'objet APDU à la méthode process() de l'applet actuellement sélectionné. L'objet APDU porte un tableau d'octet, qui contient le contenu du message d'APDU.
L'applet traite une commande APDU en appelant des méthodes dans l'objet APDU. En général, l'applet exécute les étapes suivantes:

#### Étape 1. Recherche de l'APDU buffer.

L'applet appelle la méthode getBuffer() pour obtenir une référence pour l'APDU buffer, qui contient le message. Quand l'applet reçoit l'objet APDU, seulement les cinq premiers octets de l'en-tête APDU sont disponibles dans l'APDU buffer. Ils sont l'octet CLA, INS, P1, P2, et l'octet P3 respectivement. L'octet P3 dénote l'octet LC, si la commande a des données facultatives. L'applet peut inspecter les octets d'en-tête pour déterminer la structure de la commande et l'instruction indiquée dans la commande.

#### Étape 2. Réception des données.

Si la commande APDU contient des données facultatives, l'applet doit diriger l'objet APDU pour recevoir des données entrantes en appelant la méthode setIncomingAndReceive (). Les données sont lues dans l'APDU buffer suivant les cinq bytes d'en-tête. Le dernier byte dans l'en-tête (LC) montre la longueur des données entrantes. Si l'APDU buffer ne peut pas contenir toutes les données, l'applet peut traiter les données peu à peu, ou il peut les copier dans un buffer interne. Dans l'un ou l'autre cas, il appellerait alors à plusieurs reprises la méthode receiveBytes() pour lire les données additionnelles dans l'APDU buffer.

Étape 3. Les données de retour.

Après le traitement de la commande APDU, l'applet peut également renvoyer des données au CAD dans la réponse APDU. L'applet devrait d'abord appeler la méthode setOutgoing() pour bloquer ou interdire la direction de transfert de données, et pour obtenir la longueur prévue de la réponse (Le). Cette longueur Le est indiquée dans la commande APDU appareillée avec cette réponse APDU.

Après, l'applet appelle la méthode setOutgoingLength() pour informer le CAD de la longueur réelle des données de réponse. L'applet peut déplacer les données à l'APDU buffer et appeler la méthode sendBytes() pour envoyer des données. La méthode sendBytes() peut être appelée à plusieurs reprises si l'APDU buffer ne peut pas contenir les données entières de la réponse.

Si les données sont stockées dans un buffer interne, l'applet appelle la méthode sendByteLong() pour envoyer des données du buffer.

Si les données de réponse sont assez courtes pour s'adapter dans l'APDU buffer, la classe d'APDU fournit une méthode commode pour faire ainsi: setOutgoingAndSend (). Cette méthode est une combinaison de setOutgoing, de setOutgoingLength, et de sendBytes. Cependant, cette méthode peut seulement être appelée une fois, et aucune autre méthode d'envoi ne peut être appelée après.

Étape 4. Retour des status word.

A la suite d'un retour réussi de la méthode process (), le JCRE envoie automatiquement 0x9000 pour indiquer le traitement normal. À un point quelconque, si l'applet détecte n'importe quelle erreur, l'applet peut jeter un ISOException en appelant la méthode statique ISOException.throwIt (short reason). Le mot de statut est indiqué dans le paramètre reason. Si l'ISOException n'est pas manipulé par l'applet, il sera attrapé par le JCRE. Le JCRE recherche le code complémentaire et l'envoie comme les mots de statut.

## 3.6 Conclusion

Ce chapitre fournit les informations nécessaires pour la mise au point d'un applet de la carte de Java. Comme pour le développement de n'importe quelle application software, le programmeur d'applet doit tout d'abord spécifier la fonction de l'application et désigner la structure de ses classes. En plus de ces étapes, le concepteur d'applet doit obtenir les AIDs de l'applet et du paquetage contenant l'applet, et doit spécifier les APDU commandes que l'applet aura à supporter. Une classe d'applet doit s'étendre de la classe

javacard.framework.Applet. La classe Applet définit un cadre pour l'interaction de l'applet avec le JCRE (Java Card Runtime Environment) durant le cycle de vie de l'applet. Une sous classe de la classe Applet doit outrepasser les méthodes de cette classe pour implémenter la fonction de l'applet.

A l'aide de ce chapitre, on a tout ce qu'il faut comme concepts de base de la technologie de carte Java pour pouvoir commencer à développer nos propres applets de carte.

# Chapitre4: A propos de la Biométrie

## 4.1 Introduction

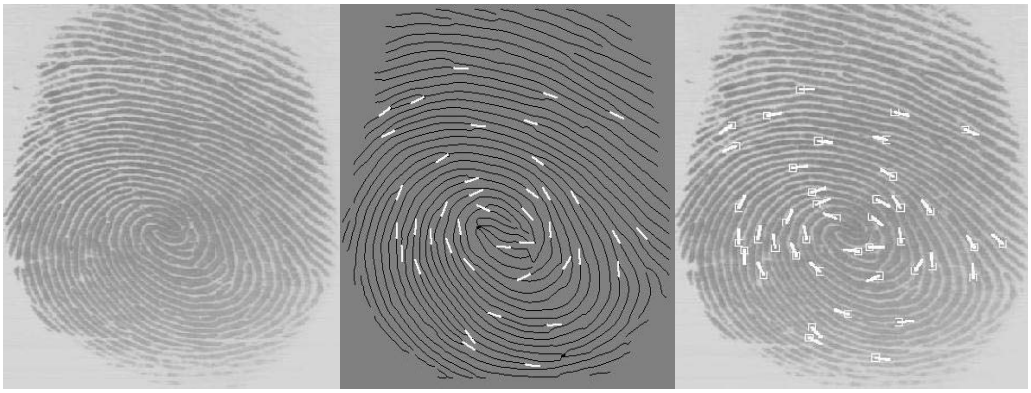
La biométrie est une méthodologie qui permet d'identifier des personnes en se basant sur leurs différentes caractéristiques physiologiques ou comportementales. Les traits biométriques incluent les empreintes digitales, la forme faciale, les modèles d'iris, les modèles de la rétine, la géométrie de la main, la parole, l'écriture et même les modèles des veines du poignet. L'identification biométrique peut être employée pour empêcher l'accès non autorisé aux bâtiments, aux bureaux, aux postes de travail, aux ordinateurs, aux serveurs, aux téléphones cellulaires, aux dispositifs sans fil, aux bases de données, et aux réseaux informatiques fermés et ouverts. La sécurité biométrique est plus robuste que d'autres méthodes telles que les mots de passe, les codes PIN ou les cartes à puce parce que la biométrie identifie des individus plutôt que des dispositifs. Ces autres méthodes de sécurité peuvent être perdues ou volées et donc être à la portée d'utilisateurs non autorisés. Un biométrique, tel qu'une empreinte digitale, est une clé qui peut ne jamais être perdue. La biométrie peut éliminer le besoin de mots de passe ou peut être employée pour consolider des mots de passe existants.

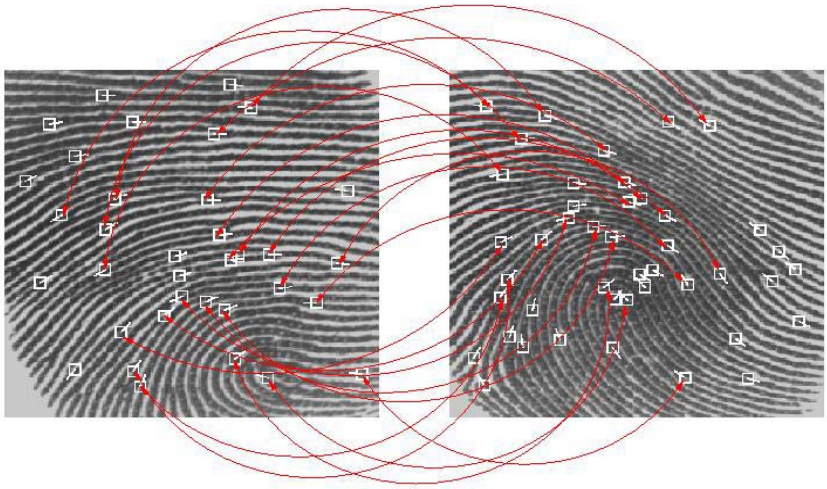
## 4.2 Assortiment d'empreinte digitale

Parmi toutes les techniques biométriques, l'identification basée sur l'empreinte digitale est la méthode la plus ancienne qui a été employée dans de nombreuses applications avec succès. Chacun est connu par ses propres empreintes digitales uniques et immuables. Une empreinte digitale est faite d'une série d'arêtes et de sillons sur la surface du doigt. L'unicité d'une empreinte digitale peut être déterminée par le modèle des arêtes et des sillons aussi bien que les points de minuties. Les points de minuties sont des caractéristiques locales d'arête qui se produisent à une bifurcation ou à une fin d'arête.

Les techniques d'assortiment d'empreinte digitale sont sous forme de deux catégories: les techniques basées sur les minuties et celles basées sur la corrélation. Les techniques basées sur les minuties trouvent tout d'abord les points de minuties et tracent ensuite leur placement relatif sur le doigt. Cependant, il y a quelques difficultés en utilisant cette approche. Il est difficile d'extraire les points de minuties exactement quand l'empreinte digitale est de mauvaise qualité. En outre cette méthode ne tient pas compte du modèle global des arêtes et les sillons. La méthode basée sur la corrélation peut surmonter certaines difficultés de l'approche basée sur les minuties. Cependant, elle a certaines de ses propres imperfections. Les techniques basées sur la corrélation exigent l'endroit précis d'enregistrement d'un point et

sont affectées par translation et rotation d'image.





L'assortiment d'empreinte digitale basé sur des minuties a des problèmes en assortissant différents modèles de minuties. Les structures locales d'arête ne peuvent pas être complètement caractérisées par des minuties. Il faut essayer une représentation alternative des empreintes digitales qui saisira plus d'information locale et rapportera un code de longueur fixe pour l'empreinte digitale. L'assortiment deviendra alors si tout va bien une tâche relativement simple de calcule de la distance euclidienne entre les deux codes.

## 4.3 Classification d'empreinte digitale

De grands volumes d'empreintes digitales sont rassemblés et stockés dans plusieurs applications telles les applications concernant la médecine, le contrôle d'accès, et l'enregistrement de permis de conducteur. Une identification automatique des personnes basée sur les empreintes digitales risque que l'empreinte digitale d'entrée soit assortie avec un grand nombre d'empreintes digitales dans une base de données (la base de données de FBI

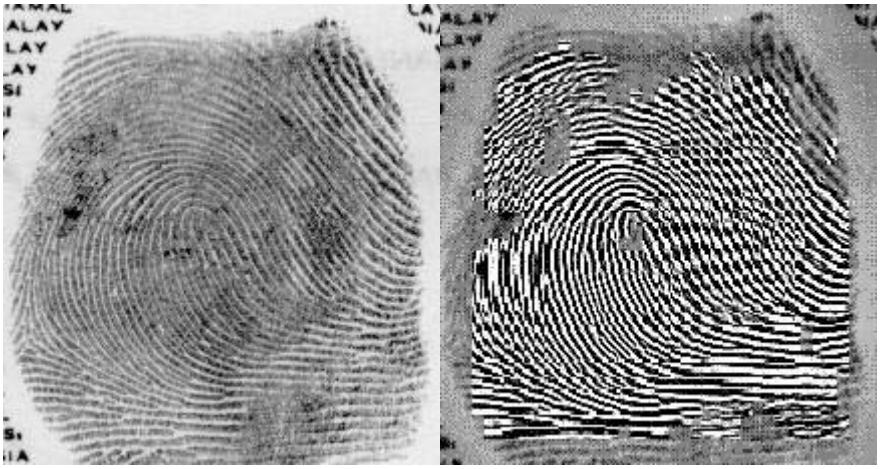
contient approximativement 70 millions d'empreintes digitales!). Pour réduire le temps de recherche et la complexité informatique, il est souhaitable de classifier ces empreintes digitales d'une façon précise et cohérente de sorte que l'empreinte digitale d'entrée soit assortie seulement avec un sous-ensemble d'empreintes digitales dans la base de données.


La classification d'empreinte digitale est une technique pour assigner une empreinte digitale dans un des multiples types pré spécifiés déjà établis dans la littérature qui peut fournir un mécanisme d'indexation. La classification d'empreinte digitale peut être regardée en tant qu'assortiment des empreintes digitales d'une façon approximative. Une empreinte digitale d'entrée est tout d'abord assortie d'une façon approximative à un des types pré spécifiés et puis, et d'une façon plus fine, elle est comparée à un sous-ensemble de la base de données contenant ce type d'empreintes digitales seulement. Un algorithme a été développé pour classifier les empreintes digitales en cinq classes, à savoir, whorl, *right loop, left loop, arch, and tented arch*. L'algorithme sépare le nombre d'arêtes en quatre directions (0 degrés, 45 degrés, 90 degrés, et 135 degrés) en filtrant la partie centrale d'une empreinte digitale avec des filtres de Gabor. Cette information est quantifiée pour produire un code pour l'empreinte servant pour la classification. La classification est basée sur un classificateur à deux étapes qui emploie un classificateur K-nearest dans la première étape et un ensemble de réseaux neurologiques dans la deuxième étape. Le classificateur est examiné sur 4.000 images dans la base de données Nist-4. Pour le problème des cinq classes, l'exactitude de classification de 90% est réalisée. Pour le problème de quatre classes (arch et tented arch combinés en une seule classe), il est possible de réaliser une exactitude de classification de 94,8%. En incorporant une option de rejet, l'exactitude de classification peut grimper à 96% pour la

classification de cinq classes et jusqu'à 97,8% pour la classification de quatre classes quand 30,8% des images sont rejetés.

## 4.4 Perfectionnement d'image d'empreinte digitale

Une étape critique dans l'assortiment automatique d'empreinte digitale est d'extraire automatiquement et sûrement des minuties à partir des images d'empreinte digitale d'entrée. Cependant, l'exécution d'un algorithme d'extraction de minuties se fonde fortement sur la qualité des images d'empreinte digitale d'entrée. Afin de s'assurer que la performance d'un système automatique d'identification/vérification d'empreinte digitale sera robuste en ce qui concerne la qualité des images d'empreinte digitale, il est essentiel d'incorporer un algorithme de perfectionnement d'empreinte digitale dans le module d'extraction de minuties. Il faut développer un algorithme rapide de perfectionnement d'empreinte digitale, qui peut de manière adaptative améliorer la clarté des structures d'arête et de sillon des images d'empreinte digitale d'entrée basées sur l'orientation et la fréquence locales estimées d'arête. Il faut évaluer la performance de l'algorithme de perfectionnement d'image en utilisant l'index de qualité des minuties extraites et l'exactitude d'un système de vérification d'empreinte digitale. Les résultats expérimentaux montrent que l'incorporation des algorithmes de perfectionnement améliore l'index de qualité et l'exactitude de vérification.



# Chapitre5: Environnement de développement de carte à puce Java

## 5.1 La compagnie GEMPLUS

Gemplus est le fournisseur principal mondial des solutions basées sur la technologie de carte à puce. Gemplus est la seule compagnie qui se consacre globalement aux cartes à puce. Avec des activités dans 37 pays, Gemplus a la plus grande capacité de production dans l'industrie et une présence dans chaque marché principal.

## 5.2 Environnement de travail : GemXpresso RAD III

### 5.2.1 Introduction

GemXpresso RADIII est le kit de développement dédié à programmer des applications spécifiques pour les cartes Java.


**Figure 12: L'interface utilisateur du GemXpresso RADIII**

Le kit GemXpresso rad III est la solution de Gemplus pour permettre à n'importe quel développeur de JavaCard de concevoir et examiner de nouvelles applications à valeur ajoutée pour des cartes de GemXpresso. Il inclut tous les éléments nécessaires requis pour assurer un procédé complet de développement dans un environnement de carte de Java.

### 5.2.2 Les bénéfices du GemXpresso RADIII

- **Une fois écrite, s'exécute n'importe où**

La norme de la carte Java " une fois écrite, s'exécute n'importe où" définit une plate-forme ouverte et interopérable où les programmeurs peuvent écrire des applets et les exécuter sur n'importe quelle carte Java, indépendamment du logiciel d'exploitation ou du circuit utilisé.

Les applets développées avec GemXpresso RADIII peuvent être chargés sur n'importe quel Carte Java comme GemXpresso211, GemXpressoLite ou GemXpressoPro.

♦ **Plus de services dans la carte**

Débit/crédit, fidélité, commerce mobile sont peu d'exemples des services qui peuvent être conçus, mis en application et examinés en utilisant l'outil de développement GemXpresso RADIII.

♦ **Flexibilité après établissement de la carte**

Les nouvelles applications et les services développés avec GemXpresso RADIII peuvent être ajoutés à votre carte de GemXpresso, ceci adapte dynamiquement vos offres de service à l'utilisateur.

♦ **Délai d'arrivée au marché plus rapide**

Maintenant, plusieurs applets peuvent résider sans risque dans la même carte à puce grâce à la machine virtuelle sécurisée et aux mécanismes du firewall de la carte Java assurant la coexistence saine et sécurisée entre les applets sur la même carte. Les services à valeur ajoutée développés par GemXpresso RAD III aident à augmenter l'utilisation de la carte, offre des services exclusifs aux clients existants. L'authentification d'Internet, le commerce électronique GemXpresso RADIII donne l'accès à la technologie de la carte à puce à la communauté large des développeurs de Java. En raison de l'intégration des outils multiples (chargeur, simulateur, programme de mise au point...) dans un environnement unifié, la programmation est plus facile et plus efficace.

### 5.2.3 Contenu

Le kit de GemXpresso RAD III fournit tous les outils requis pour assurer un procédé complet de développement dans un environnement de carte de Java. Il contient plusieurs composants de logiciel pour développer, charger, corriger et examiner des applications off-card et des applets de carte de Java:

- **Wizards et templates:** Aident un développeur à concevoir rapidement l'applet de la carte de Java et les bibliothèques nécessaires ou à importer un projet existant. magiciens
- Un **plugin** disponible en JBuilder et café visuel:

Permet d'éditer le projet en cours de la carte de Java, lance la conversion ou tous les autres outils.

- **Le simulateur de GemXpresso de la carte à puce :**

Tous les simulateurs peuvent être utilisés dans tous les programmes de mise au point standards (JBuilder, VCafe,..). il est employé pour examiner l'applet de la carte de Java, pour insérer le point d'arrêt ou pour le corriger point par point. Il peut être employé dans un debbuger ou avec son vue graphique interne.

- Une interface graphique utilisateur **GUI** (Graphical User Interface) appelée **JcardManager :**

Permet d'envoyer différentes commandes ou APDU à la carte à puce. Il offre la possibilité de charger des applet, de contrôler la carte à puce, de jouer ou d'enregistrer le manuscrit.

### 5.2.4 Les spécifications techniques du GemXpresso RAD III

GemXpresso RAD III est entièrement conforme avec la majorité des normes tel que la carte de Java 2.1, la plate-forme ouverte 2.0.1, cadre ouvert de carte ou PCSC.

Il fonctionne sous :

- Windows 95/98
- Windows NT Workstation
- Windows 2000
- Linux Red Hat 6.

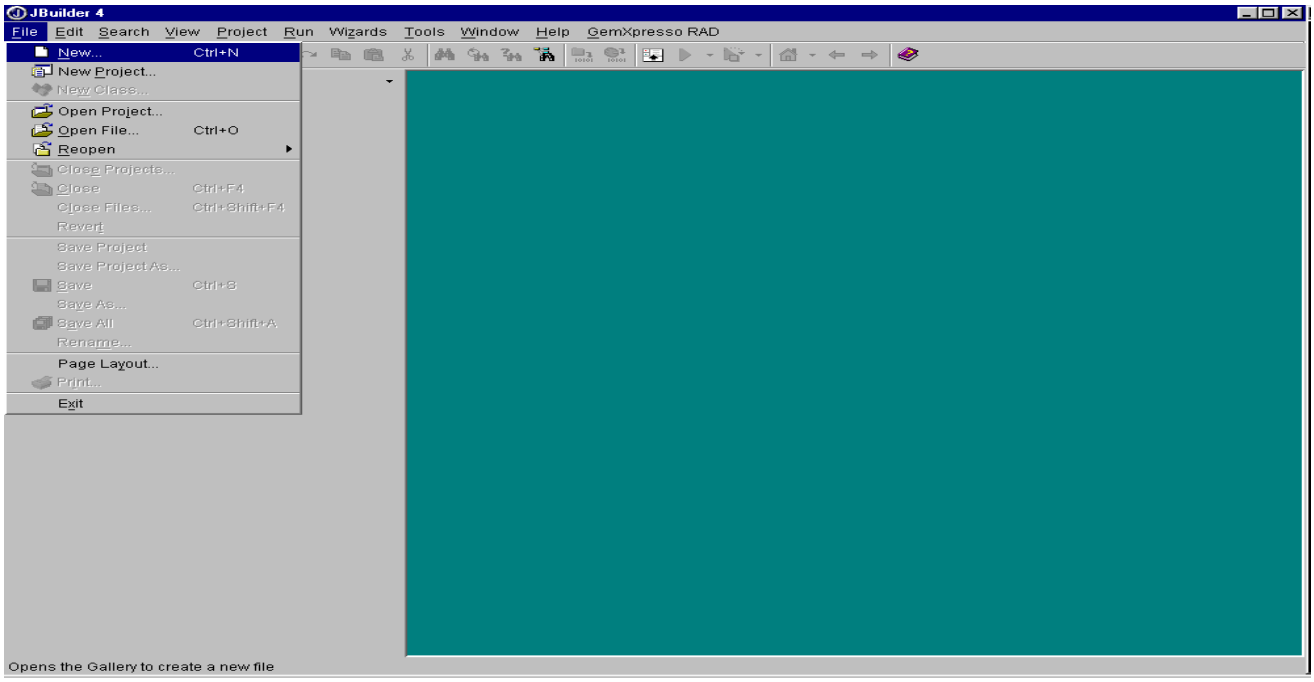
### 5.2.5 Savoir se servir du GemXpresso RAD III

**a- Création d'un projet avec Jbuilder**

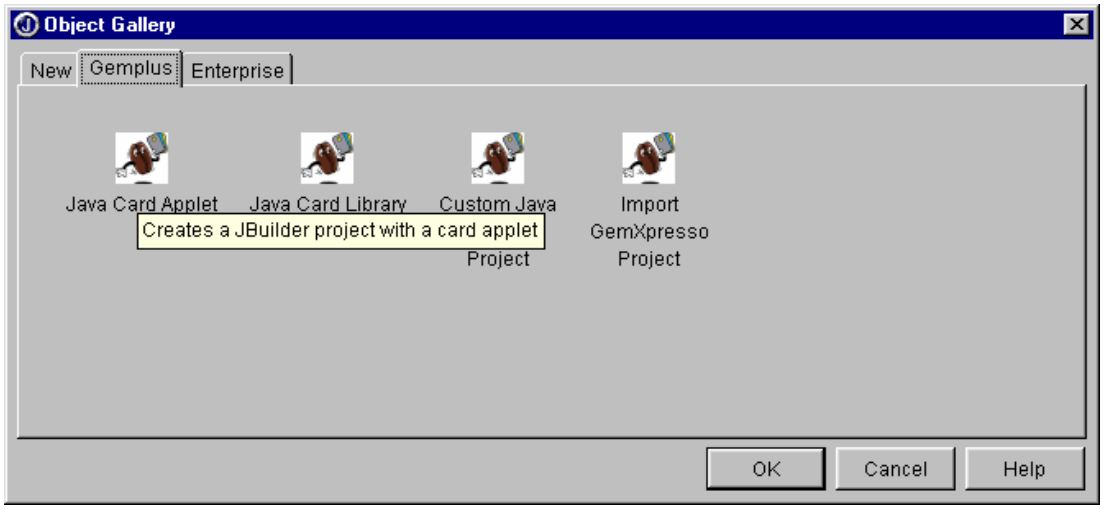
Comme c'est déjà mentionné, le logiciel GemXpresso RAD III sert à inscrire des données dans la carte à puce Java. Pour ce fait, on va se servir de ce logiciel pour créer un applet qui permet de retourner l'expression ''HELLO WORLD'' à travers une commande APDU et la charger dans la carte pour pouvoir communiquer avec par le biais du JCard Manager.

Tout d'abord, on crée un nouveau projet comme suit :

1. On lance **JBuilder**.
2. On choisit **File>New**.

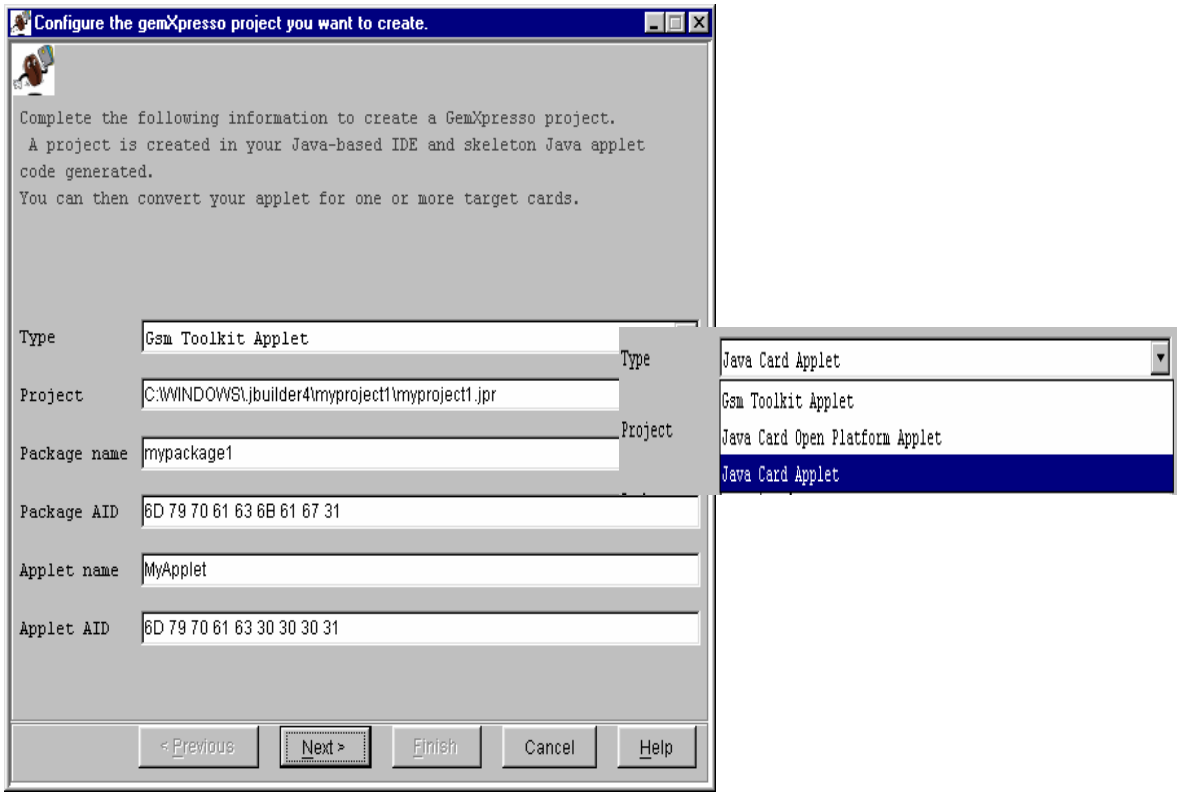



3. une fenêtre **Object Gallry** s'ouvre contenant trois étiquettes. On clique sur l'étiquette **Gemplus** :

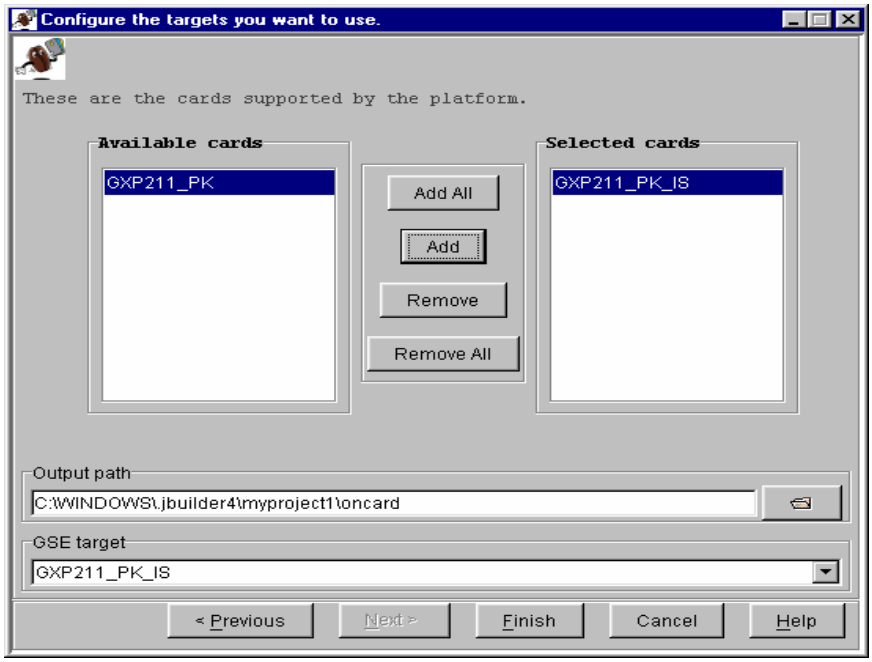



4. On sélectionne **Java Card Applet** et on clique sur **OK.**

En créant un nouveau applet de carte à puce Java, la fenêtre wizard suivante est affichée.




A cet étape, on sélectionne le type compatible avec la carte à puce Java 2.1 et le chemin menant au projet.De plus, on fait entrer le nom et le code **AID** de l'applet et le nom et le code **AID** du paquetage.Après et en cliquant sur **Next** la deuxième page de la fenêtre wizard s'affiche :



Sur cette fenêtre on configure le type la de carte cible pour lequel les dossiers de l'applet seront par la suite convertis. On choisi le type de carte **GXP211_PK_IS** et on clique sur le bouton **Add** qui transfère ce type de carte vers la liste des cartes selectionnées.
Dans le domaine **Output path**, on tape le répertoire là où les dossiers convertis du projet vont être placés. Pour annoncer la fin du **wizard** on clique **Finish.**

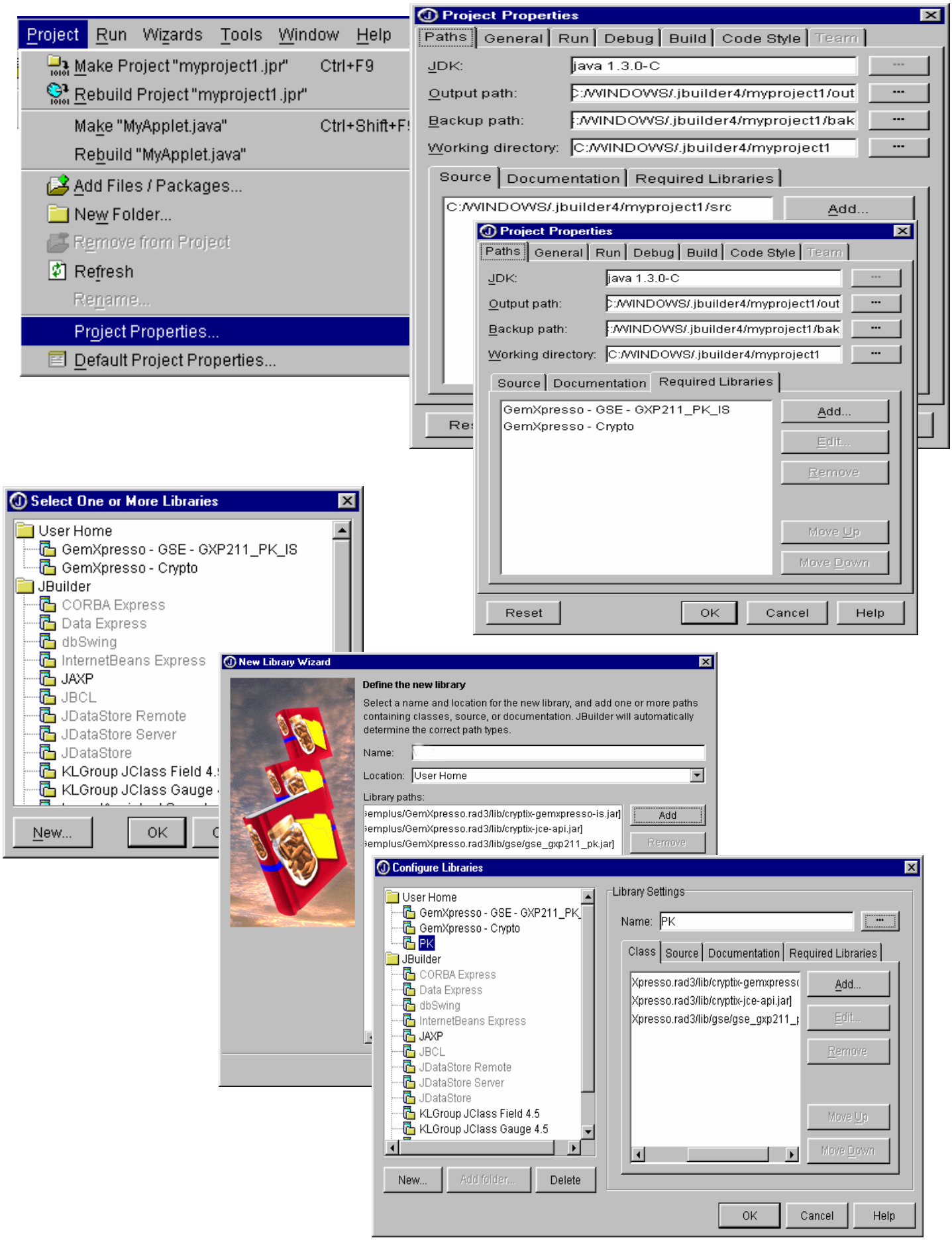
**b- Construction du projet par JBuilder**

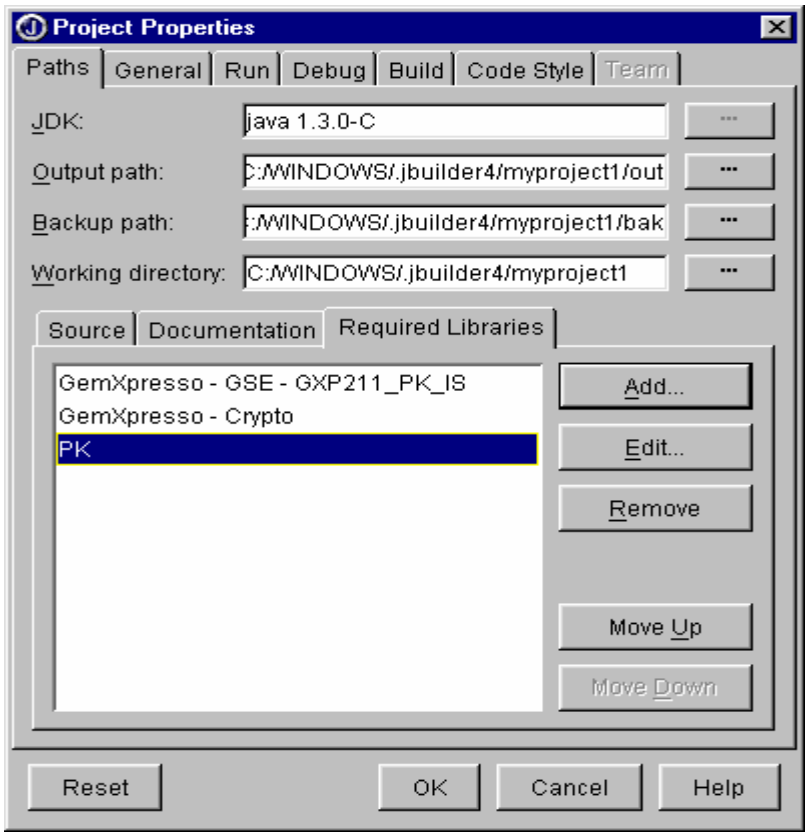
Une fois que le wizard est fini, un projet JBuilder est crée et activé. Pour construire ce projet on doit lui ajouter les bibliothèques et les chemins d'accès aux dossiers du projet.

- Une bibliothèque est créée sous la forme d'un projet de **JBuilder** ou de **visual café**, auquel on assigne un nom et un code de paquetage. Après, on ajoute au projet les classes de la bibliothèque pour les convertir ensuite vers la carte cible.

Les fichiers suivants des bibliothèques sont configurés **(properties>paths>required librairies) :**

- **GemXpresso-GSE-GXP211_PK_IS (**gse_gxp211_pk.jar**).**
- **GemXpresso-Crypto (**cryptix-jce-api.jar et cryptix-gemxpresso.jar ou cryptix-gemxpresso-is.jar).


Project Run Wizards Tools Window Help
Make Project "myproject1.jpr" Ctrl+F9
Rebuild Project "myproject1.jpr"
Make "MyApplet.java" Ctrl+Shift+F
Rebuild "MyApplet.java"
Add Files / Packages...
New Folder...
Remove from Project
Refresh
Rename...
Project Properties...
Default Project Properties...
Project Properties
Paths General Run Debug Build Code Style Team
JDK: java 1.3.0-C
Output path: C:/WINDOWS/.jbuilder4/myproject1/out
Backup path: C:/WINDOWS/.jbuilder4/myproject1/bak
Working directory: C:/WINDOWS/.jbuilder4/myproject1
Source Documentation Required Libraries
C:/WINDOWS/.jbuilder4/myproject1/src
Add...
Project Properties
Paths General Run Debug Build Code Style Team
JDK: java 1.3.0-C
Output path: C:/WINDOWS/.jbuilder4/myproject1/out
Backup path: C:/WINDOWS/.jbuilder4/myproject1/bak
Working directory: C:/WINDOWS/.jbuilder4/myproject1
Source Documentation Required Libraries
GemXpresso - GSE - GXP211_PK_IS
GemXpresso - Crypto
Add...
Edit...
Remove
Move Up
Move Down
Reset OK Cancel Help
Select One or More Libraries
User Home
GemXpresso - GSE - GXP211_PK_IS
GemXpresso - Crypto
JBuilder
CORBA Express
Data Express
dbSwing
InternetBeans Express
JAXP
JBCL
JDataStore Remote
JDataStore Server
JDataStore
KLGroup JClass Field 4.5
KLGroup JClass Gauge 4.5
New... OK
New Library Wizard
Define the new library
Select a name and location for the new library, and add one or more paths containing classes, source, or documentation. JBuilder will automatically determine the correct path types.
Name:
Location: User Home
Library paths:
emplus/GemXpresso.rad3/lib/cryptix-gemxpresso-is.jar]
emplus/GemXpresso.rad3/lib/cryptix-jce-api.jar]
emplus/GemXpresso.rad3/lib/gse/gse_gxp211_pk.jar]
Add
Remove
Configure Libraries
User Home
GemXpresso - GSE - GXP211_PK
GemXpresso - Crypto
PK
JBuilder
CORBA Express
Data Express
dbSwing
InternetBeans Express
JAXP
JBCL
JDataStore Remote
JDataStore Server
JDataStore
KLGroup JClass Field 4.5
KLGroup JClass Gauge 4.5
New... Add folder... Delete
Library Settings
Name: PK
Class Source Documentation Required Libraries
Xpresso.rad3/lib/cryptix-gemxpresso
Xpresso.rad3/lib/cryptix-jce-api.jar]
Xpresso.rad3/lib/gse/gse_gxp211_
Add...
Edit...
Remove
Move Up
Move Down
OK Cancel Help




- Le répertoire **output path (Project>Project Properties>Paths >OutputPath)** est placé au **projectdir/out**, où **projectdir** est le nom du répertoire qu'on spécifie pour le dossier du projet.

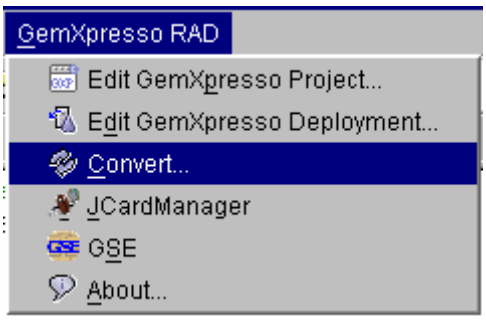
GemXpresso RADIII crée automatiquement des sous-répertoires au-dessous de ce répertoire et stocke les fichiers Classe dans le répertoire approprié, conformément aux conventions de Java.

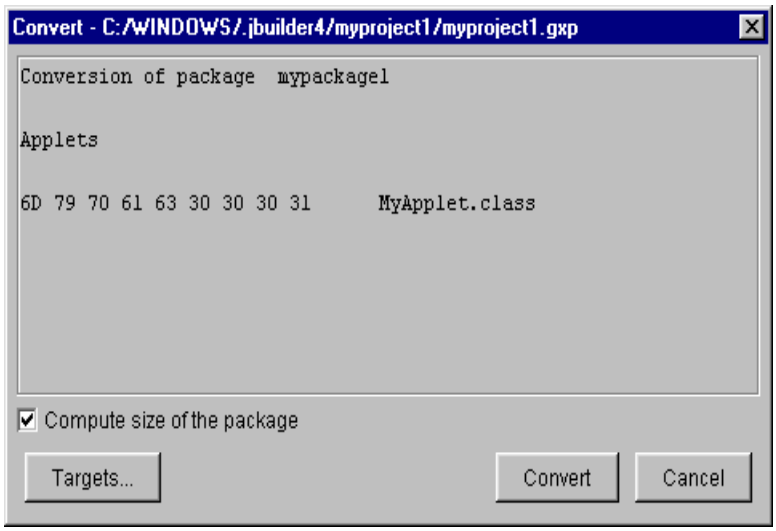
- Le répertoire fichier source **(Project>Project Properties>Paths >Source)** est placé au **projectdir/src**, où **projectdir** est le nom du répertoire qu'on spécifie pour le nom du projet.

GemXpresso RADIII crée automatiquement des sous-répertoires au-dessous de cette répertoire et stocke les fichiers Classe dans le répertoire approprié, conformément aux conventions de Java.

### c- Conversion des fichiers

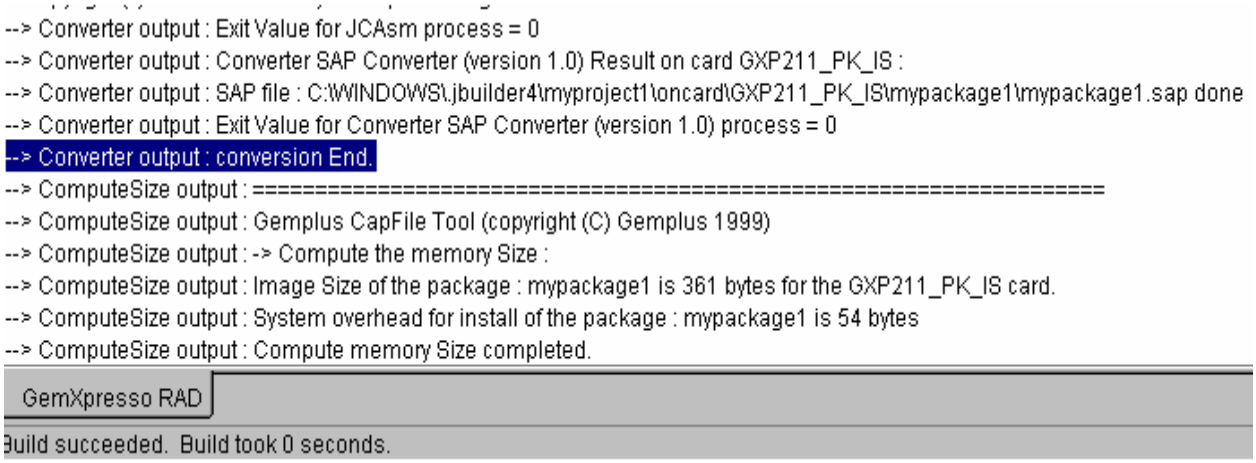
On utilise le **GxpConverter utility** pour convertir les fichiers classe du paquetage en des fichiers de format **CAP**(pour la carte) ou de format **SAP**(pour le simulateur **GSE**) respectivement nécessaires pour le chargement dans la carte ou dans le simulateur.




Pour convertir les fichiers de l'applet, on choisit **GemXpressoRAD>Convert** pour afficher la fenêtre **Convert.** Pour commencer la conversion on clique le bouton **Convert.**

Les messages de conversion sont affichés progressivement dans la fenêtre des messages de l'IDE (environnement de développement intégré) :




Le **GxpConverter** crée pour chaque type de carte les fichiers suivants :

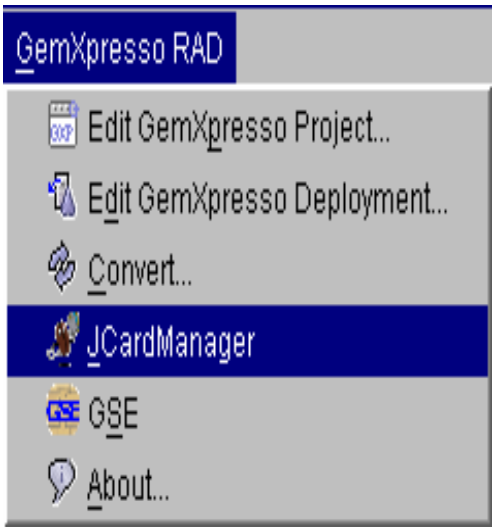
- Un fichier Java Archive (.JAR) contenant les fichiers CAP (.CAP) prêts à être chargés dans la carte.
- Un fichier SAP (.SAP) prêt à être chargé dans le GSE (le simulateur de la carte).

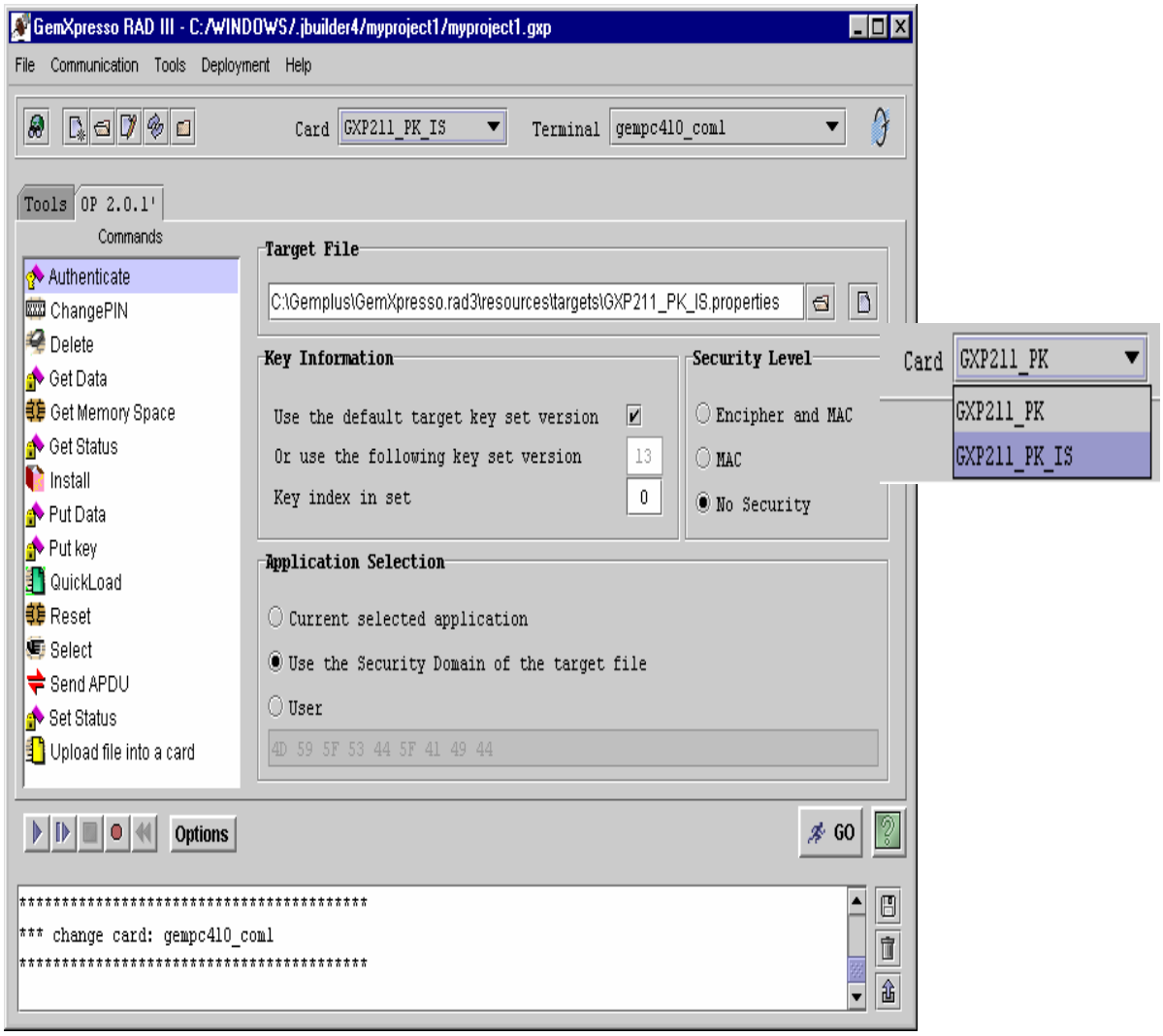
- Un fichier export (.EXP).
- Un fichier JCA (.JCA).

Ces fichiers seront placés dans le répertoire qu'on a déjà spécifié comme chemin de sortie.

**d- Lancement du JCard Manager**

Pour lancer le **Jcard Manager**, on doit choisir **Start>Programs>Gemplus Applications>GemXpresso PAD III>Jcard Manager.**La fenêtre **Jcard Manager** s'affiche :



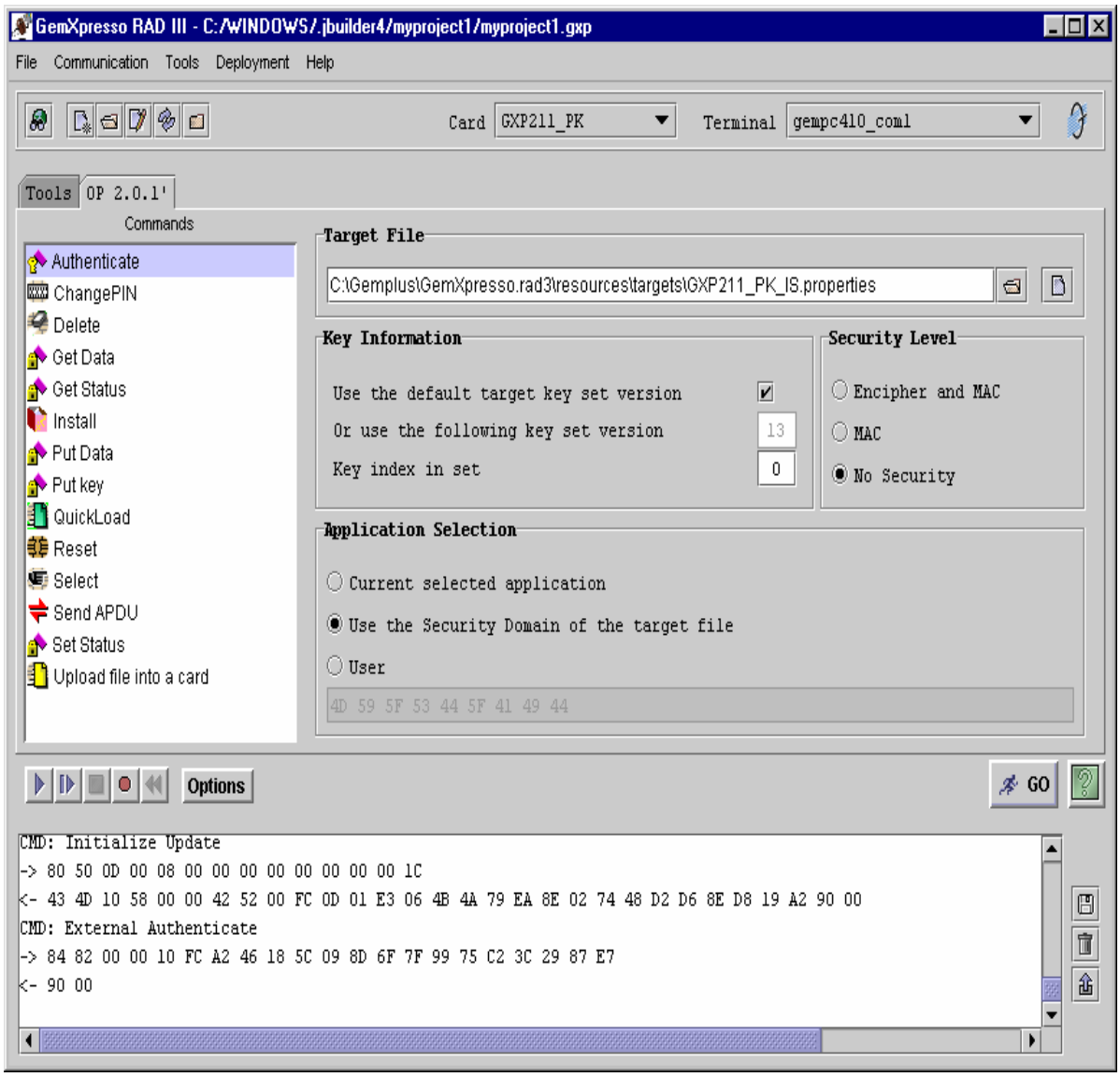




Noter bien que le **JCard Manager** détecte automatiquement si c'est le simulateur ou bien le lecteur de carte **gempc410_com** qui est connecté, pour afficher Simulateur respectivement **gempc410_com** dans la liste Terminal de sa fenêtre.

Dans la fenêtre du Jcard Manager, on choisi **GXP211_PK_IS** comme la carte cible.

La fenêtre Jcard Manager doit alors afficher le type de carte et le terminal désirés.

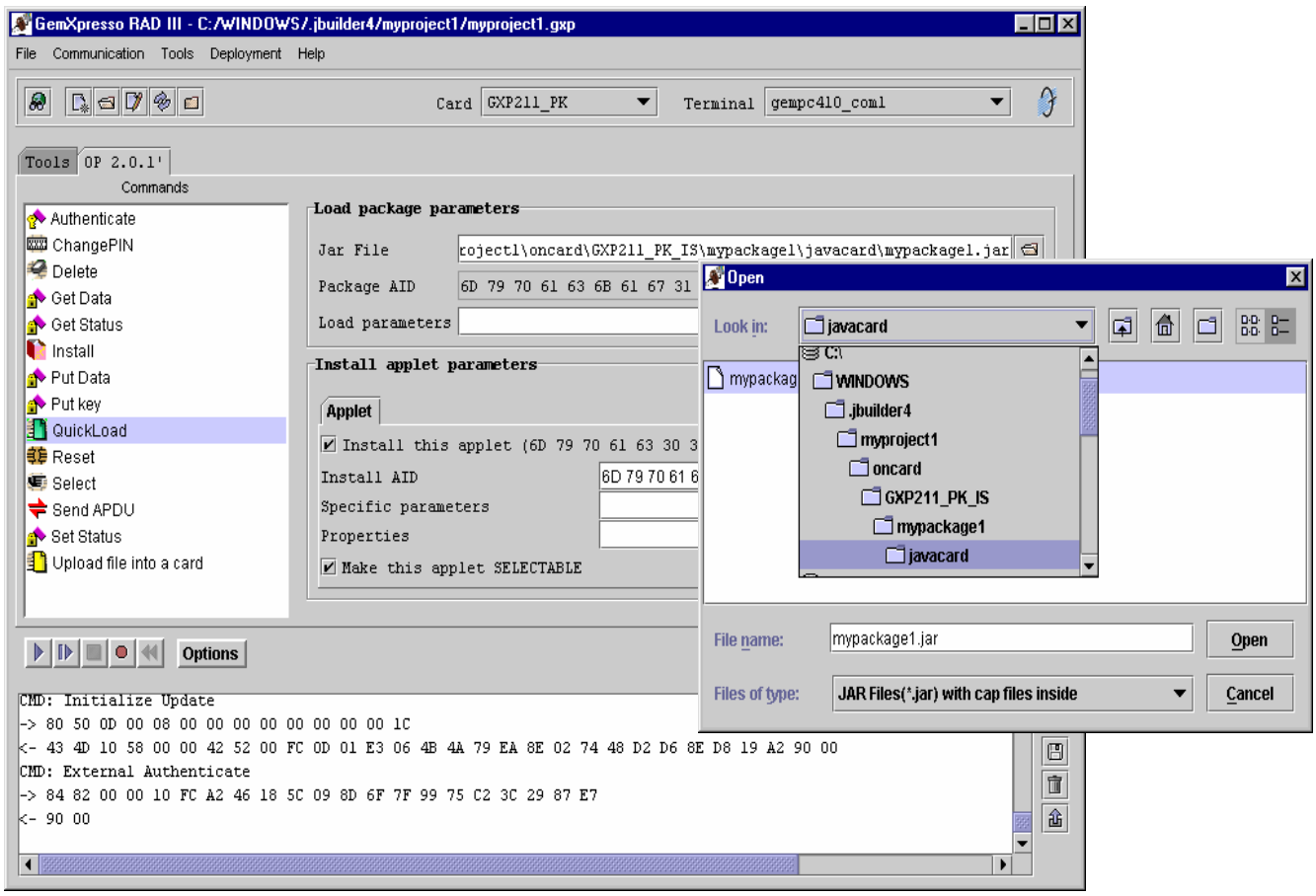
### e- Authentification de la carte




La première étape en toute session de transmission entre une entité **off-card** (typiquement une application client comme le Jcard Manager) et la carte est d'entreprendre une authentification mutuelle. Après une authentification, la carte contrôle que les clés dans le fichier principal apparient ses propres et l'entité **off-card** vérifie que les clés sur la carte apparient ses propres clés enregistrées dans le fichier principal.

Après l'authentification réussie, une voie de transmission sécurisée est établie entre les deux entités, le long desquelles d'autres commandes peuvent être envoyées.

### f- Choix de l'applet pour être chargé et installé dans la carte




Pour charger l'applet dans la carte, on lance le **JCard Manager** et on s'assure que le lecteur de carte est bien sélectionné dans cette fenêtre. Ensuite, et dans la liste **OP 2.0.1 Commands**, on sélectionne la commande **Quickload.** Cette commande permet de charger et d'installer l'applet.

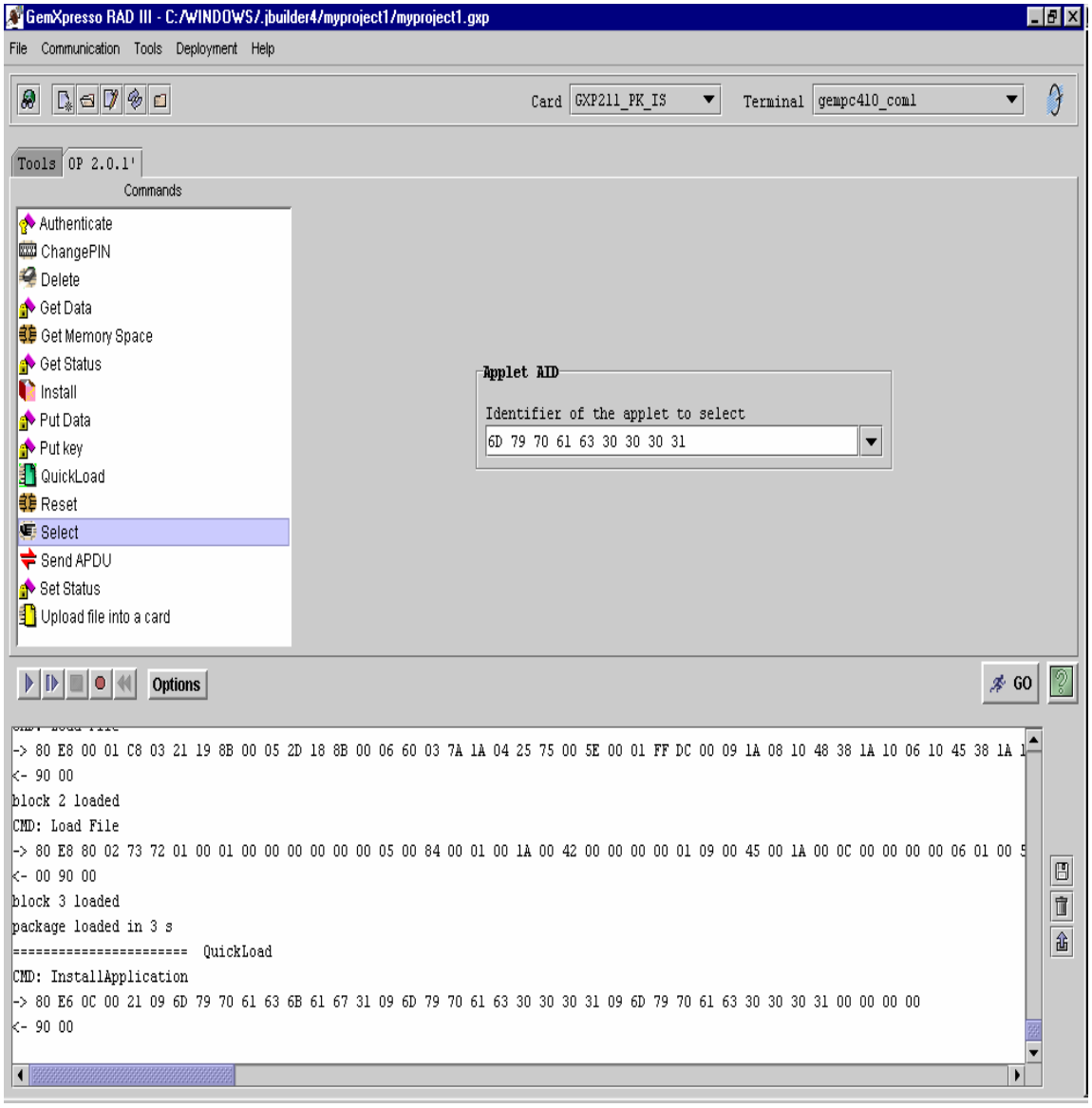
Dans le cadre **Input file**, on clique sur le bouton parcourir pour sélectionner le fichier **JAR** à charger et installer en même temps.

Dans le cadre **Package AID**, on fait entrer le code **AID** de l'applet.

Pendant que le chargement continue, les messages de statut sont sans interruption affichés dans la zone message de la fenêtre de JCard Manager. les dernières lignes indiquent que le module a été téléchargé avec succès.

### g- Sélection de l'applet

Pour sélectionner l'applet après avoir lancer le **JCard Manager**, on choisit de la liste de commandes **OP 2.0.1**, la commande **Select.** Ensuite, on tape le paquetage **AID** de l'applet et on clique **Go** pour envoyer la commande à la carte. Dans le cas du simulateur, le résultat de la commande est affiché simultanément dans la zone de messages du Jcard Manager ainsi que dans la fenêtre de la carte simulateur.

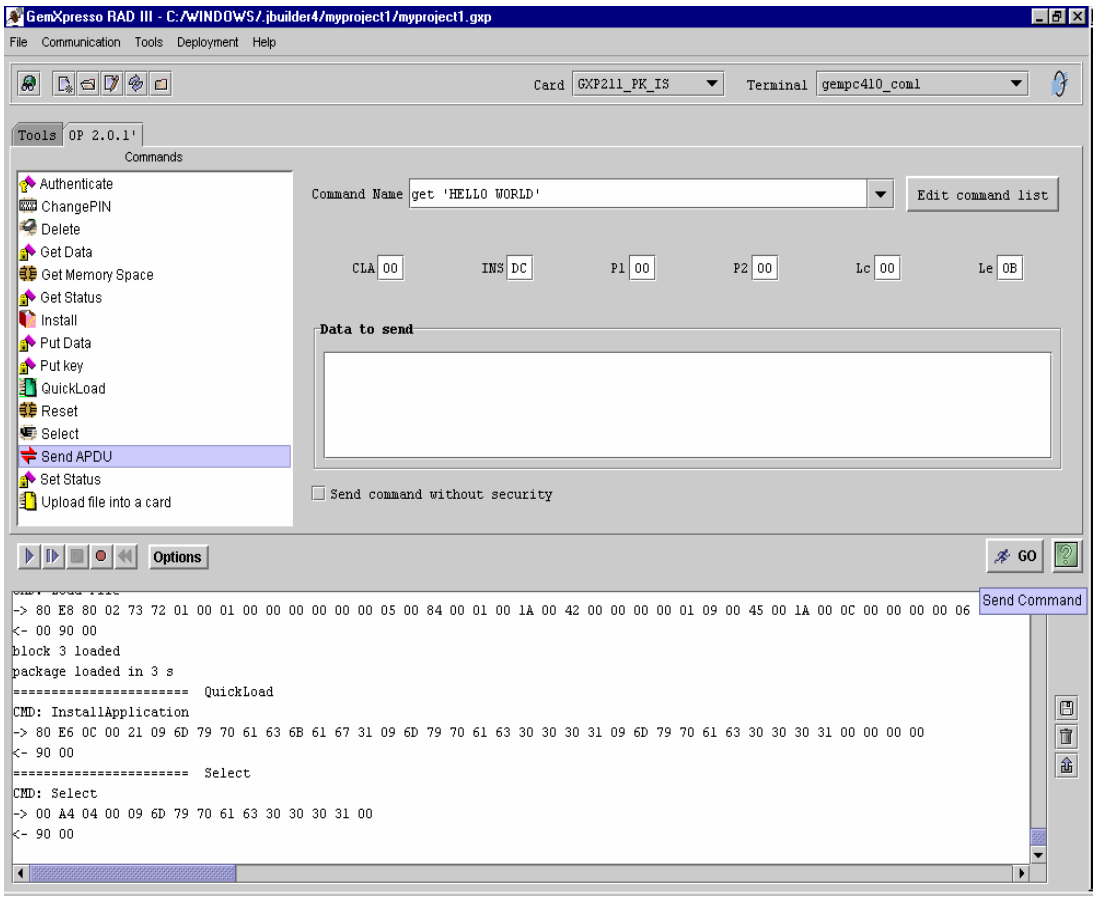


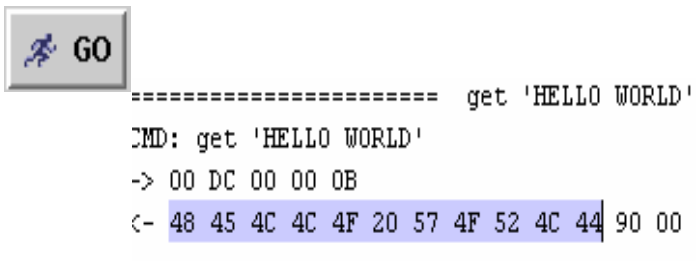

### h- Exécution de l'APDU ''HELLO WORLD ''

La commande **APDU** 'HELLO WORLD' ou getHelloWorld correspondant aux paramètres **APDU (CLA ; INS ; P1 ; P2 ; Lc, Le)** déjà définis dans l'applet permet de retourner l'expression HELLO WORLD qui sera affichée dans la zone message de la Jcard Manager.

Pour envoyer cette commande **APDU** (application protocol data unit), il suffit de sélectionner à partir de la liste de commandes **OP2.0.1** la commande **Send APDU**, de taper le nom de la commande dans la case **command name** , de faire entrer les paramètres **APDU** dans leurs cases correspondantes et cliquer le bouton **Go**.

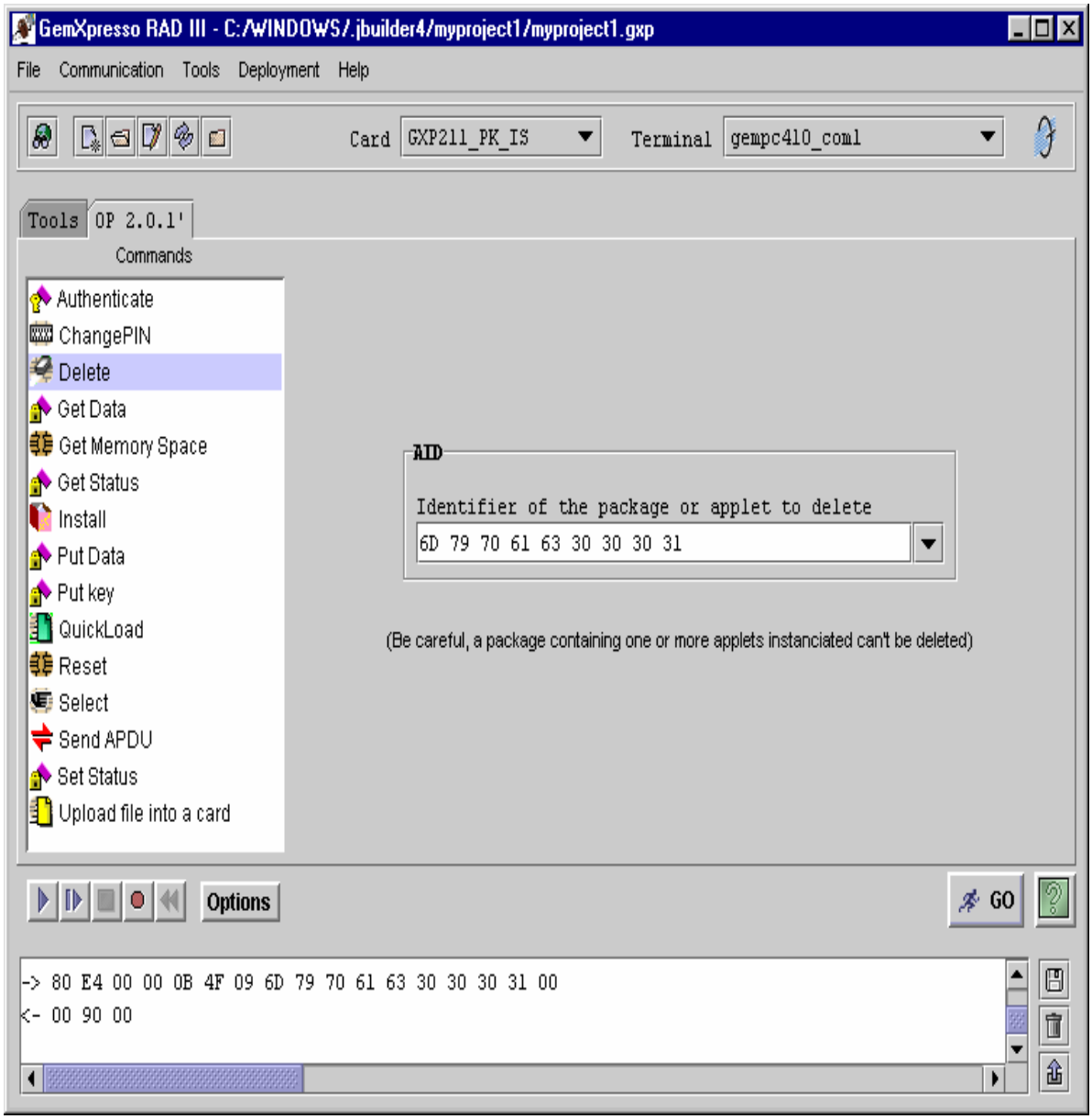
Le résultat de la commande sera affiché en code hexadécimal suivi du code de succès ''90 00''.



### i- Effacement du paquetage et de l'applet

Pour effacer l'applet de la carte, on doit sélectionner à partir de la liste des commandes OP2.0.1, la commande **Delete**. Dans la case **identifier of the package or applet to delete**, on tape tout d'abord le code **AID** de l'applet suivi bouton Go ; ensuite, on tape le code **AID** du paquetage suivi encore du bouton **Go.** Cet ordre doit être respecté pour recevoir dans la zone de messages le code de succès.




## 5.3 Explications et exécutions des fonctions de quelques programmes

### 5.3.1 Les fonctions de vérification, de crédit et de débit

**a- L'applet Wallet**

Le programme qu'on va étudier, effectue la fonction d'un porte-monnaie électronique, c'est-à-dire qu'on peut créditer ou débiter le montant actuel et visualiser le montant disponible. Par ailleurs, pour débuter ces opérations, il faut entrer un code PIN. D'après le programme, il existe quatre instructions possibles : la vérification du code PIN pour accéder aux différentes applications comme créditer ou débiter un montant. Ainsi que, l'instruction GET_BALANCE qui permet de visualiser le montant actuel. En outre, une seule vérification est suffisante pour réaliser un débit ou un crédit.

Tout d'abord, on va décrire chaque partie du programme. Ensuite, on exposera la simulation du programme tout en débitant par le contrôle du code PIN pour réaliser par la suite un crédit et un débit. Après, chacune de ces deux précédentes opérations, nous allons visualiser ce qu'il reste dans le porte-monnaie avec la méthode GET_BALANCE.

**b- Code source de l'applet et explications**

Nom du paquetage

**package mypackage;**

**import javacard.framework.*;**

**public class Wallet extends Applet {**

Déclaration des constantes pour les commandes APDU.

Déclaration de la constante CLA.

**final static byte Wallet_CLA =(byte)0xB0;**

Déclaration des différentes instructions de la constante INS.

**final static byte VERIFY = (byte) 0x20;**

**final static byte CREDIT = (byte) 0x30;**

**final static byte DEBIT = (byte) 0x40;**

**final static byte GET_BALANCE = (byte) 0x50;**

Valeur maximum de la variable balance.

**final static short MAX_BALANCE = 0x7FFF;**

Valeur maximum du montant à débiter ou à créditer.

**final static byte MAX_TRANSACTION_AMOUNT = 127;**

Nombre d'essai possible pour le code PIN avant que le PIN soit bloqué.

**final static byte PIN_TRY_LIMIT =(byte)0x03;**

Taille maximum du code PIN

**final static byte MAX_PIN_SIZE =(byte)0x08;**

Déclaration du signal caractérisant un mauvais code PIN.

**final static short SW_VERIFICATION_FAILED =0x6300;**

Déclaration d'un signal caractérisant un code PIN correct pour un crédit ou un débit.

**final static short SW_PIN_VERIFICATION_REQUIRED = 0x6301;**

Déclaration d'un signal signalant l'erreur du choix du montant.

Le montant doit être inférieur à un seuil et positif.

=> amount > MAX_TRANSACTION_AMOUNT or amount < 0.

**final static short SW_INVALID_TRANSACTION_AMOUNT = 0x6A83;**

Déclaration d'un signal lorsque la variable balance excède un maximum.

**final static short SW_EXCEED_MAXIMUM_BALANCE =0x6A84;**

Déclaration d'un signal lorsque la variable balance est négative.

**final static short SW_NEGATIVE_BALANCE = 0x6A85;**

Initialisation du code PIN.

**final static byte[] myPin={0,0,0,0};**

Déclaration des variables.

**OwnerPIN pin;**

**short balance;**

**Début du constructeur**

**private Wallet (byte[] bArray,short bOffset,byte bLength){**

C'est mieux d'attribuer à l'applet toute la mémoire dont il a besoin durant sa durée de vie dans le constructeur.

**pin = new OwnerPIN(PIN_TRY_LIMIT, MAX_PIN_SIZE);**

Les paramètres d'installation contiennent la valeur d'initialisation du PIN**.**

**pin.update(bArray, bOffset, bLength);**

**register();**

**}** fin du constructeur.

**Début de la méthode install**

**public static void install(byte[] bArray, short bOffset, byte bLength){**

Création d'une instance d'applet wallet.

**new Wallet(myPin, (short) 0, (byte) 4);**

**}** fin de la méthode install.

**Début de la méthode select**

**public boolean select() {**

L'applet refuse d'être choisi si le code PIN est bloqué.

**if ( pin.getTriesRemaining() == 0 )**

**return false;**

**return true;**

}fin de la méthode select.

**Début de la méthode deselect**

**public void deselect() {**

Réinitialisation du code PIN.

**pin.reset();**

}fin de la méthode deselect.

**Début de la méthode process**

**public void process(APDU apdu) {**

L'objet APDU contient le tableau d'octets et transfère les données entre la carte et le lecteur. Seuls les premier octets [CLA, INS, P1, P2, LC] sont disponibles dans le buffer APDU. L'interface javacard.framework.ISO7816 déclare les constantes de ces octets dans le buffer APDU.

L'applet appelle la méthode getBuffer pour obtenir une référence de l'APDU buffer.

**byte [] buffer = apdu.getBuffer();**

Vérification de la commande APDU SELECT.

**if ((buffer[ISO7816.OFFSET_CLA] == 0) &&**

**(buffer[ISO7816.OFFSET_INS] == (byte)(0xA4)) )**

**return;**

Vérification de la commande CLA qui spécifie la structure de la commande.

**if (buffer[ISO7816.OFFSET_CLA] != Wallet_CLA)**

**ISOException.throwIt**

**(ISO7816.SW_CLA_NOT_SUPPORTED);**

**switch (buffer[ISO7816.OFFSET_INS]) {**

Appel de la méthode GET_BALANCE

**case GET_BALANCE: getBalance(apdu);**

**return;**

appel de la méthode DEBIT

**case DEBIT: debit(apdu);**

**return;**

appel de la méthode CREDIT

**case CREDIT: credit(apdu);**

**return;**

appel de la méthode VERIFY

**case VERIFY: verify(apdu);**

```
                return;
default:    ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
  }
} Fin de la méthode process.
```

**Méthodes privées**

**Début de la méthode crédit**

```
private void credit(APDU apdu) {
Vérification du code pin.
if ( ! pin.isValidated() )
ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);
```

L'applet appelle la méthode getBuffer pour obtenir une référence de l'APDU buffer. Les données APDU sont disponibles dans 'apduBuffer'.

```
byte[] buffer = apdu.getBuffer();
```

L'octet Lc correspond au nombre d'octets dans le champs de données entrantes de la commande APDU.

```
byte numBytes = buffer[ISO7816.OFFSET_LC];
```

Réception des données entrantes.

=> Indiquez que cet APDU a des données entrantes et recevez les données à partir de l'excentrage ISO7816.OFFSET_CDATA suivant les 5 bytes d'en-tête.

```
byte byteRead = (byte)(apdu.setIncomingAndReceive());
```

C'est une erreur si le nombre de bytes de données lus ne correspond pas au nombre de ceux dans le byte de LC.

```
if ( ( numBytes != 1 ) || (byteRead != 1) )
ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
```

Obtention du montant du crédit.

```
byte creditAmount = buffer[ISO7816.OFFSET_CDATA];
```

Vérification du montant du crédit.

Le montant du crédit doit être inférieur à un seuil et ne doit pas être négatif sinon une exception apparaît.

```
 if ( ( creditAmount > MAX_TRANSACTION_AMOUNT) || ( creditAmount < 0 ) )
ISOException.throwIt(SW_INVALID_TRANSACTION_AMOUNT);
```

Vérification de la nouvelle balance.

La nouvelle balance doit être inférieure à un seuil sinon une exception apparaît.

**if ( (short)( balance + creditAmount) > MAX_BALANCE )**

**ISOException.throwIt (SW_EXCEED_MAXIMUM_BALANCE);**

Créditer le montant.

**balance = (short)(balance + creditAmount);**

} Fin de la méthode crédit.

**Début de la méthode débit.**

**private void debit(APDU apdu) {**

Vérification du code pin.

**if ( ! pin.isValidated() )**

**ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);**

L'applet appelle la méthode getBuffer pour obtenir une référence de l'APDU buffer. Les données APDU sont disponibles dans 'apduBuffer'.

**byte[] buffer = apdu.getBuffer();**

**byte numBytes = (byte)(buffer[ISO7816.OFFSET_LC]);**

**byte byteRead = (byte)(apdu.setIncomingAndReceive());**

Si les octets numbytes ou byteRead sont différents de 1 alors il y a levée d'une exception.

**if ( ( numBytes != 1 ) || (byteRead != 1) )**

**ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);**

Obtention du montant du débit.

**byte debitAmount =buffer[ISO7816.OFFSET_CDATA];**

Vérification du montant du débit. Le montant du débit doit être inférieur à un seuil et il doit être positif, sinon une exception apparaît.

**if ( ( debitAmount > MAX_TRANSACTION_AMOUNT) || (debitAmount < 0))**

**ISOException.throwIt(SW_INVALID_TRANSACTION_AMOUNT);**

Vérification de la nouvelle balance. La nouvelle balance doit être toujours positive sinon une exception apparaît.

**if ( (short)( balance - debitAmount ) < (short)0 )**

**ISOException.throwIt(SW_NEGATIVE_BALANCE);**

**balance = (short) (balance - debitAmount);**

} Fin de la méthode débit.

### Début de la méthode getbalance

**private void getBalance(APDU apdu) {**

L'applet appelle la méthode getBuffer pour obtenir une référence de l'APDU buffer. Les données APDU sont disponibles dans 'apduBuffer'.

**byte[] buffer = apdu.getBuffer();**

Le système est informé que l'applet a fini le traitement de la commande et le système doit à nouveau se préparer à construire une réponse APDU qui contient le champs de données.

**short le = apdu.setOutgoing();**

**if ( le < 2 )**

**ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);**

Le lecteur est informé du nombre réel de bytes retourné.

**apdu.setOutgoingLength((byte)2);**

Déplacement des données de la balance dans le buffer APDU à partir de l'offset 0.

**buffer[0] = (byte)(balance >> 8);**

**buffer[1] = (byte)(balance & 0xFF);**

Envoie des deux bytes de la variable balance à l'offset 0 dans l'APDU buffer.

**apdu.sendBytes((short)0, (short)2);**

} Fin de la méthode getbalance.

### Début de la méthode verify

**private void verify(APDU apdu) {**

L'applet appelle la méthode getBuffer pour obtenir une référence de l'APDU buffer. Les données APDU sont disponibles dans 'apduBuffer'.

**byte[] buffer = apdu.getBuffer();**

Recherche des données du PIN pour la validation.

**byte byteRead = (byte)(apdu.setIncomingAndReceive());**

Vérification du PIN.

Les données du PIN sont lues dans l'APDU buffer à l'offset ISO7816.OFFSET_CDATA.

**if ( pin.check(buffer, ISO7816.OFFSET_CDATA, byteRead) == false)**

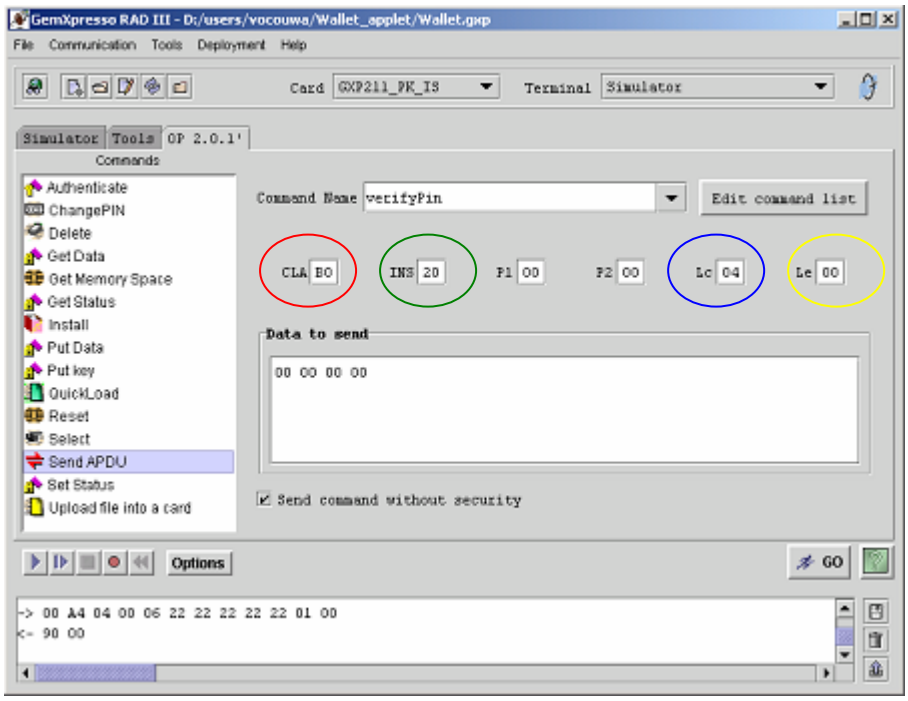
**ISOException.throwIt(SW_VERIFICATION_FAILED);**

} fin de la méthode verify.

} fin de la class wallet.

### c- Exécution du programme

### 1. Vérification du code PIN :




CLA décrit la classe d'instruction indiquant la structure et le format pour une catégorie de commande APDU. D'après le code, la constante a été déclarée comme (byte) 0xB0.

```
// code of CLA byte in the command APDU header
  final static byte Wallet_CLA =(byte)0xB0;
```

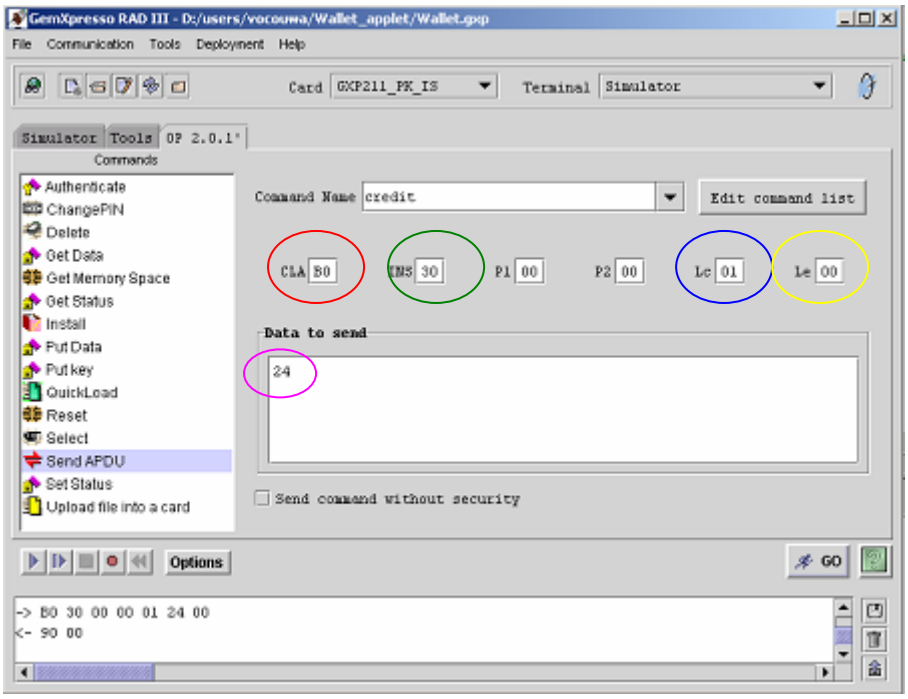
INS représente l'instruction de la commande. Par ailleurs, il en existe quatre dont celle-ci (vérification du code PIN) qui est déclarée comme (byte) 0x20.

```
// codes of INS byte in the command APDU header
final static byte VERIFY = (byte) 0x20;
```

Lc indique la longueur des données de commande.

Le indique la longueur prévue des données de réponse. Dans ce cas, aucune réponse n'est attendue.

### 2. Crédit d'un montant :




CLA décrit la classe d'instruction indiquant la structure et le format pour une catégorie de commande APDU. D'après le code, la constante a été déclarée comme (byte) 0xB0.

L'instruction correspondant au crédit est déclarée comme (byte) 0x30.

```
// codes of INS byte in the command APDU header
final static byte CREDIT = (byte) 0x30;
```

D'après le programme, si l'octet Lc est différent de 1 alors une erreur se produit.

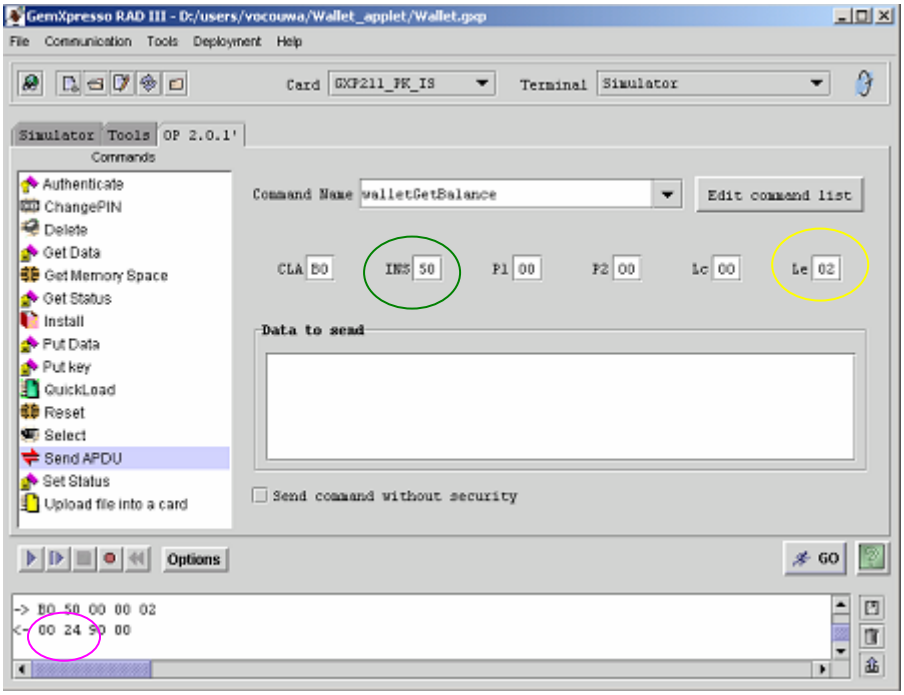
```
byte numBytes = buffer[ISO7816.OFFSET_LC];
byte byteRead =(byte)(apdu.setIncomingAndReceive());
// it is an error if the number of data bytes
// read does not match the number in Lc byte
if ( ( numBytes != 1 ) || (byteRead != 1) )
ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
```

En ce qui concerne l'octet Le, aucune valeur n'est attendue.

A cet endroit, on peut mentionner le montant désiré (ici 24).

Dans le cas suivant, on va contrôler le montant actuel.

### 3. Contrôle du montant actuel :




**A présent comme une valeur doit être retournée, l'octet Le indique la longueur prévue des données de réponse.**

```
short le = apdu.setOutgoing();
if ( le < 2 )
ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
//informs the CAD the actual number of bytes
//returned
apdu.setOutgoingLength((byte)2);
```

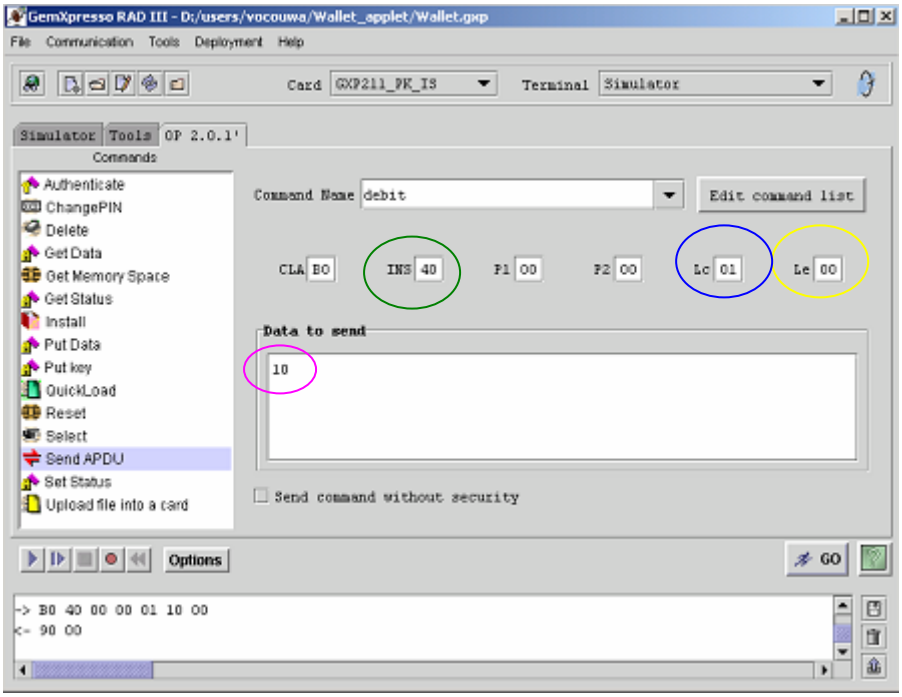
A cet emplacement, on peut constater qu'il s'agit bien de la valeur 24 qui a été prise en compte.

**Cette instruction est déclarée comme (byte) 0x50.**

```
// codes of INS byte in the command APDU header
final static byte GET_BALANCE = (byte) 0x50;
```

Maintenant, nous pouvons débiter le montant :

### 4. Débit d'un montant :




Le montant qu'on désire retirer est 10.

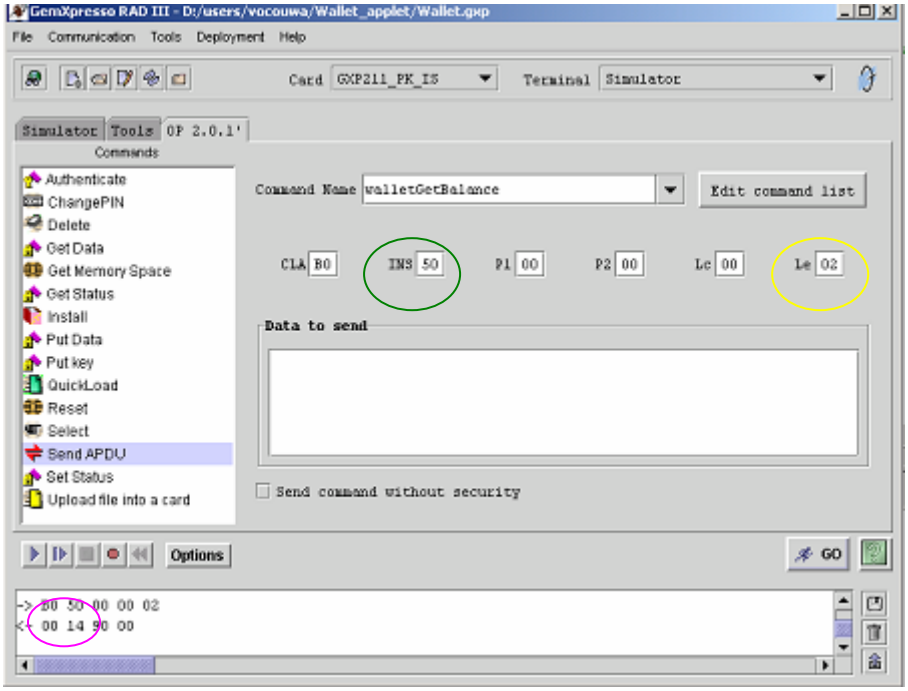
En ce qui concerne l'octet Le, aucune valeur n'est attendue.

D'après le programme, si l'octet Lc est différent de 1 alors une erreur se produit.

L'instruction débit est déclarée comme (byte) 0x40.

```
// codes of INS byte in the command APDU header
final static byte DEBIT = (byte) 0x40;
```

### 5. Contrôle du montant actuel après avoir débité:




A présent comme une valeur doit être retournée, l'octet Le indique la longueur prévue des données de réponse.

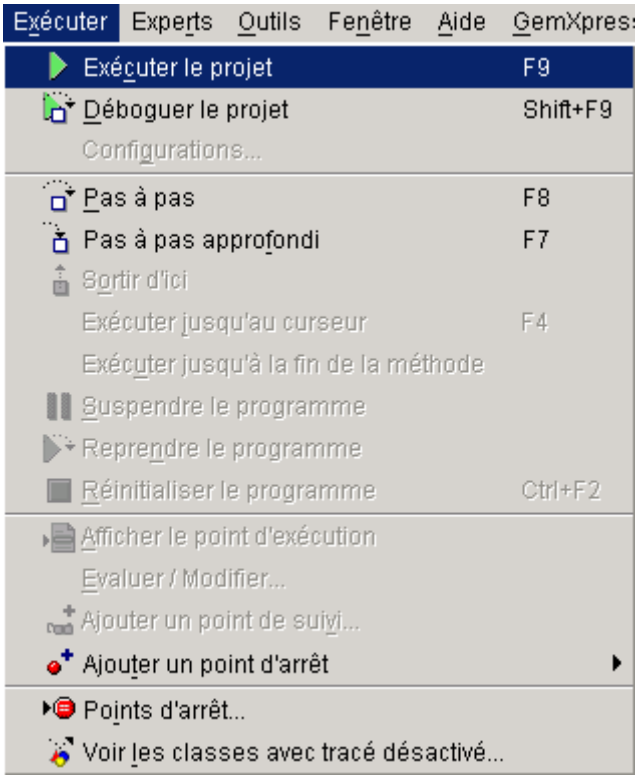
A cet emplacement, on peut constater qu'il s'agit bien de la valeur 14, puisque le montant était de 24 préalablement et qu'on a retiré 10.

Cette instruction est déclarée comme (byte) 0x50.

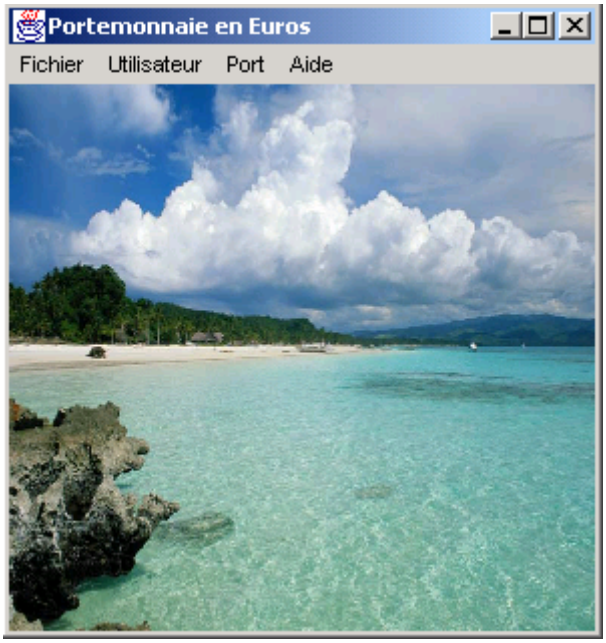
### 5.3.2 Les fonctions d'un porte-monnaie électronique : Basique

**a- Exécution de l'applet Basique client**

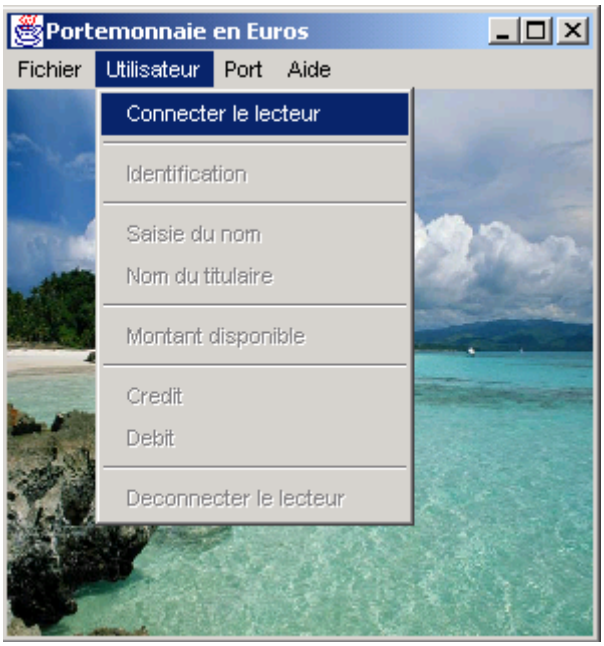
Le programme nommé 'Basique client' réalise un porte-monnaie électronique. Pour accéder à la vérification des codes PIN permettant de mettre à jour ou d'afficher le nom du propriétaire de la carte et de créditer ou de débiter un montant, on doit exécuter le programme. Il faut bien noter que lors de la simulation, la carte ne doit pas être dans le lecteur.



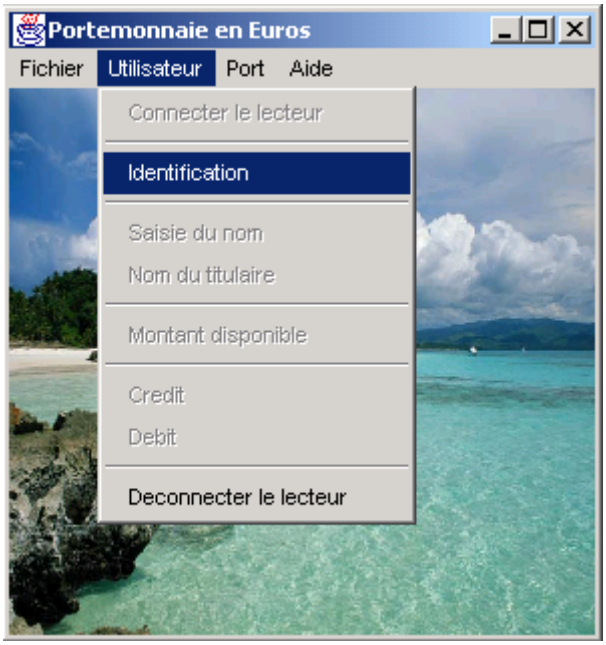

Après l'exécution du programme, l'image suivante apparaît faisant office du porte-monnaie électronique.




Tout d'abord, on doit connecter le lecteur.




Une fois que la connexion du lecteur est établie, on doit s'identifier en tapant le code PIN.

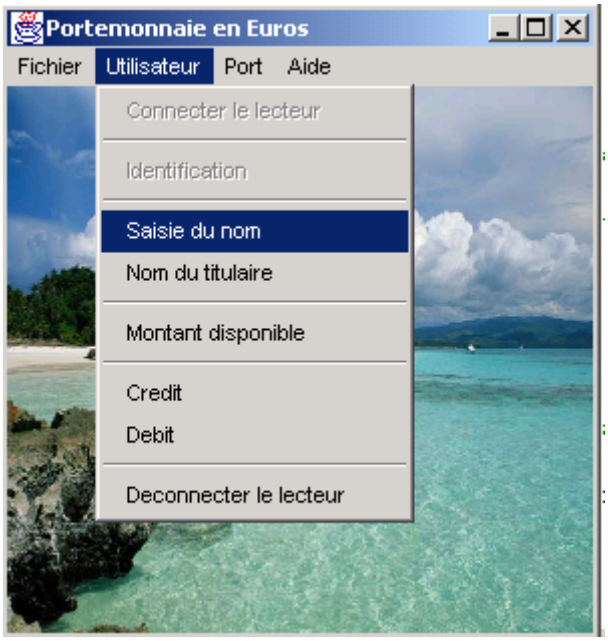


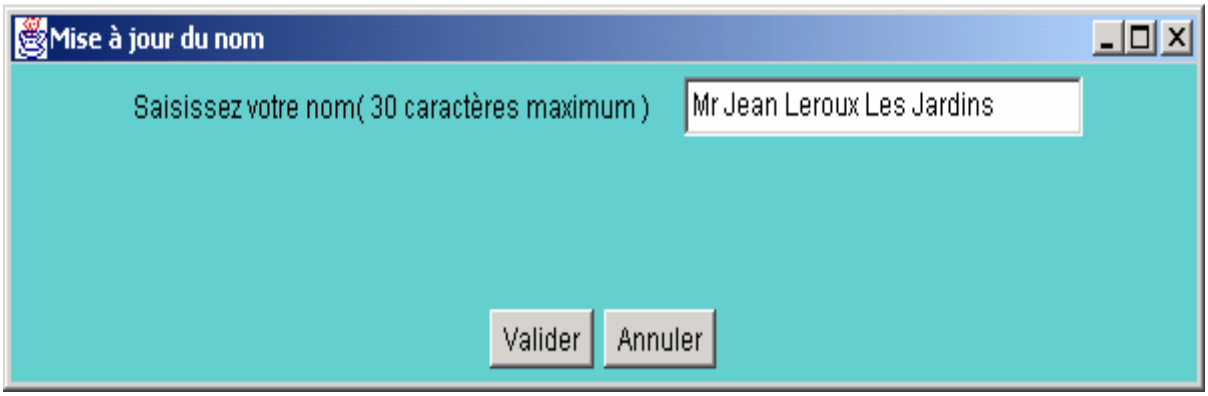

D'après le programme, le code PIN permettant l'accès à la mise à jour ou à l'affichage du nom du titulaire, aux crédits ou débits et aux visualisations des montants, est '1234' :

**System.out.println(« Initialize OP global PIN ») ;**
**System.out.println(«») ;**
**PINResult pinResult = serv.changePIN(« 1234 », 6, null) ;**


**Figure 13: Saisie du code PIN**

Une fois que le bon code est tapé et validé, une nouvelle liste s'affiche. Pour stocker le nom du titulaire sur la carte, il suffit de cliquer sur la commande **Saisie du nom** qui à son tour ouvre une nouvelle fenêtre pour y placer le nom.

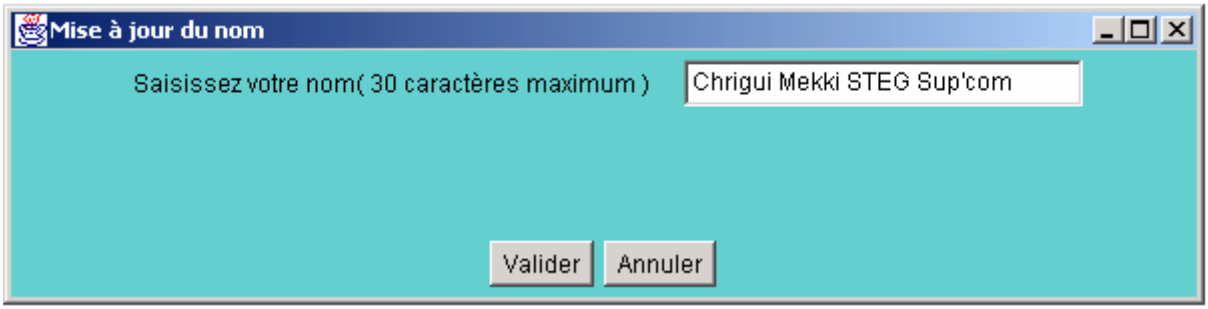



On peut bien vérifier l'existence du nom dans la carte en lui faisant appel à l'aide de la commande **nom du titulaire** qui permet d'afficher une fenêtre contenant le nom déjà stocké.



le nom entré dans la carte n'est pas définitif, il peut être modifié autant de fois qu'on veut. Pour cela, il suffit de recommencer à nouveau les mêmes étapes que précédemment et les deux fenêtres de stockage et d'affichage seront à nouveau ouvertes. Comme par exemple **:**




De la même manière, on peut créditer la carte à condition que le montant disponible ne dépasse pas 10000 Euros.

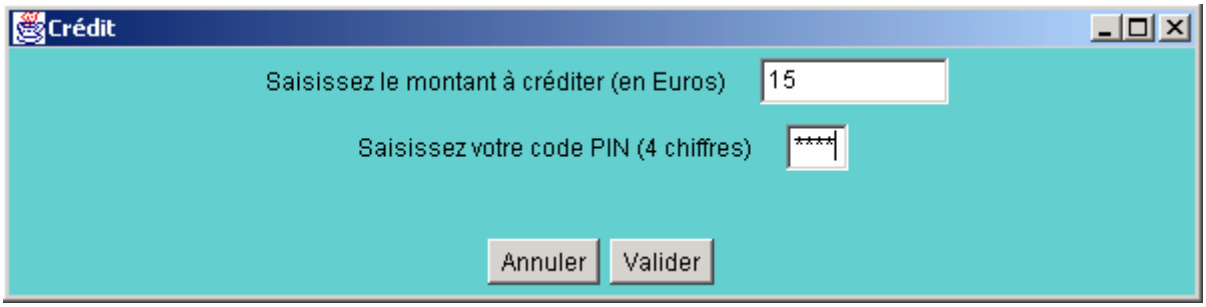

Le code PIN qui permet suite à sa validation de créditer la carte est déclaré dans le programme :

**private final static byte [] pin_credit**

**{(byte)0x30, (byte)0x30, (byte)0x30, (byte)0x30}**

(En code hexadécimal, '0x30' correspond à '0').

Le code PIN pour créditer un montant est fixé dans l'applet à '0000'. Dans notre exemple, on créditera le montant de 15 Euros :

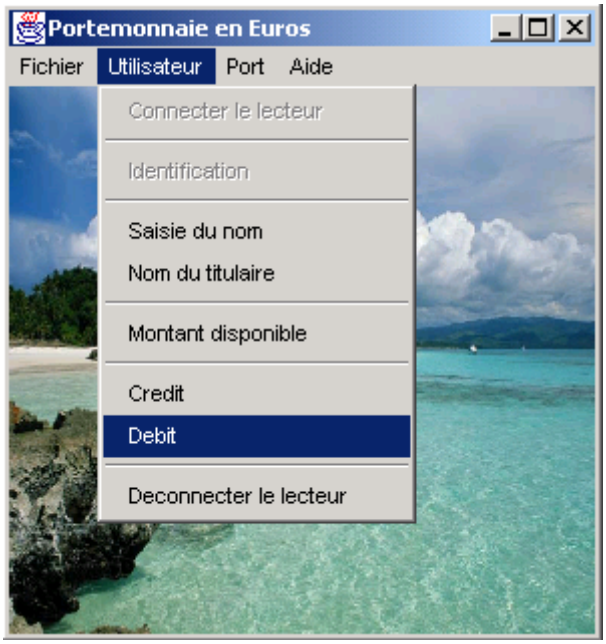


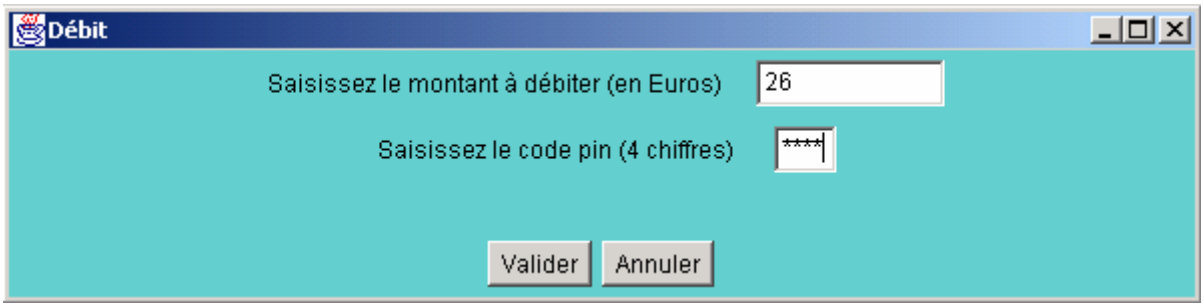
A présent, on visualise le nouveau montant:


Le code PIN qui permet suite à sa validation de débiter la carte est déclaré dans le programme:

**private final static byte [] pin_debit**
**{(byte)0x31, (byte)0x31, (byte)0x31, (byte)0x31}**

(En code hexadécimal, '0x31' correspond à '1').
Le code PIN pour débiter un montant est fixé dans l'applet à '1111'. Dans notre exemple, on réalisera un débit de 26 Euros :







A présent, le montant disponible est de 89 Euros. Il est à signaler qu'on ne peut pas débiter plus que le montant déjà existant.

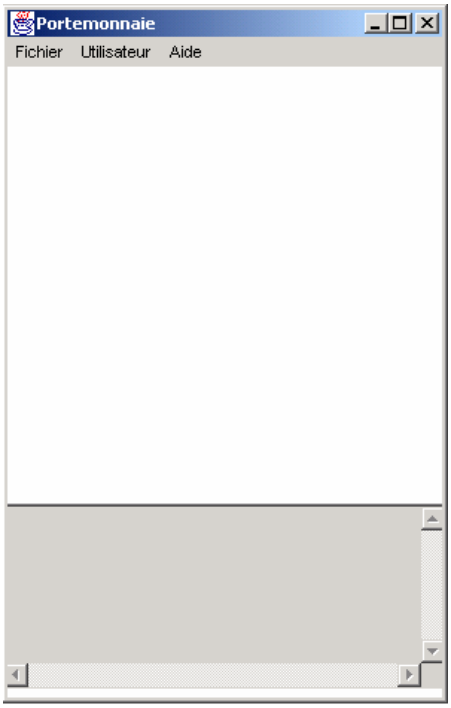
Enfin, après avoir communiquer avec la carte et qu'on veut fermer l'application, il suffit de cliquer sur la commande Déconnecter le lecteur.

### 5.3.3 Exécution de l'application money client

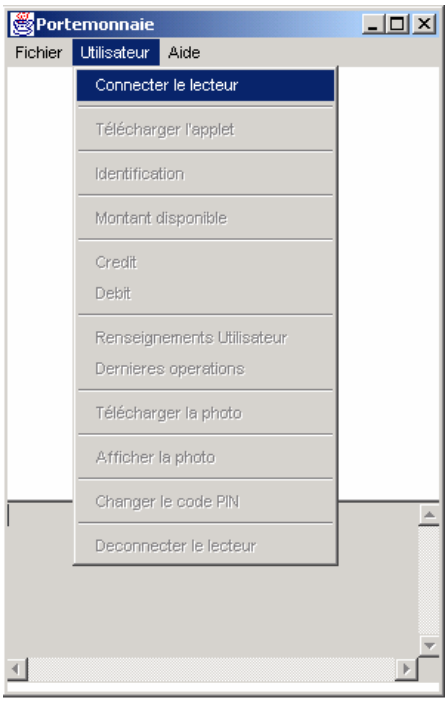
Ce programme réalise la fonction d'un porte-monnaie électronique et permet aussi la visualisation d'une photographie. Pour cette exécution, on a besoin de deux programmes qui font communiquer la carte et le terminal.

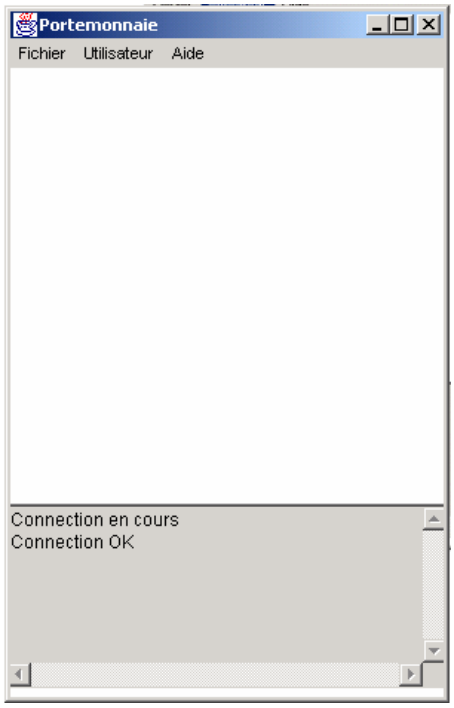
Ces deux programmes sont l'applet money-applet stockée dans la carte et l'application client lui faisant appel et servant comme interface graphique pour cette exécution. De cette façon on aura pas besoin à utiliser le JCardManager lors de l'exécution. En ce qui concerne ce programme, une seule photo peut être téléchargée.

En exécutant le projet, la fenêtre suivante apparaît:

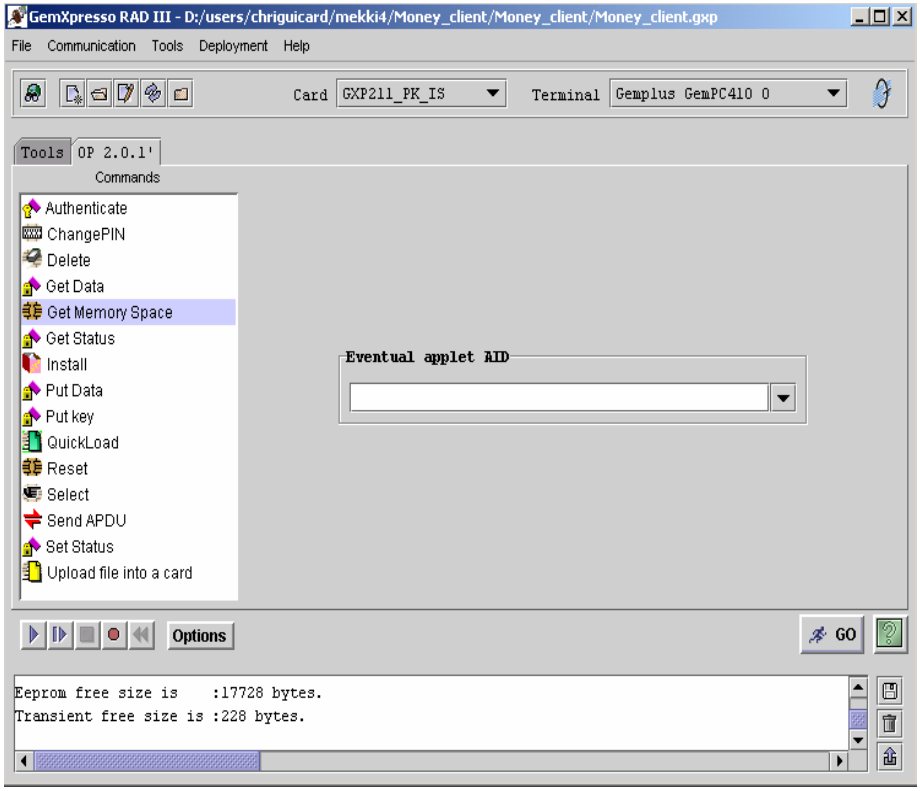



Pour connecter le lecteur on sélectionne comme d'habitude de la liste des commandes la commande **connecter le lecteur:**



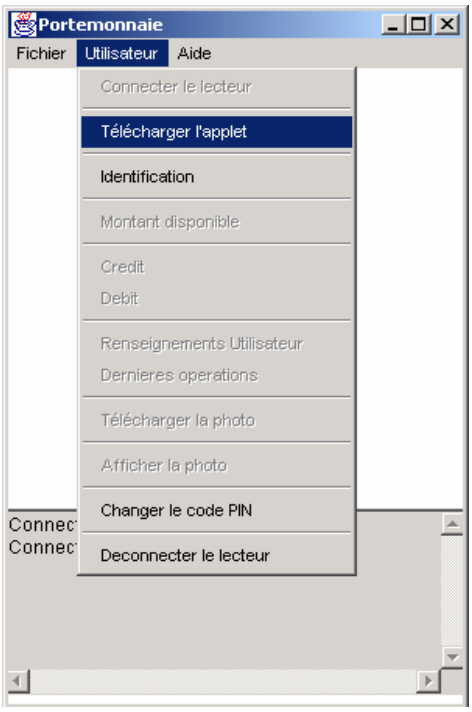



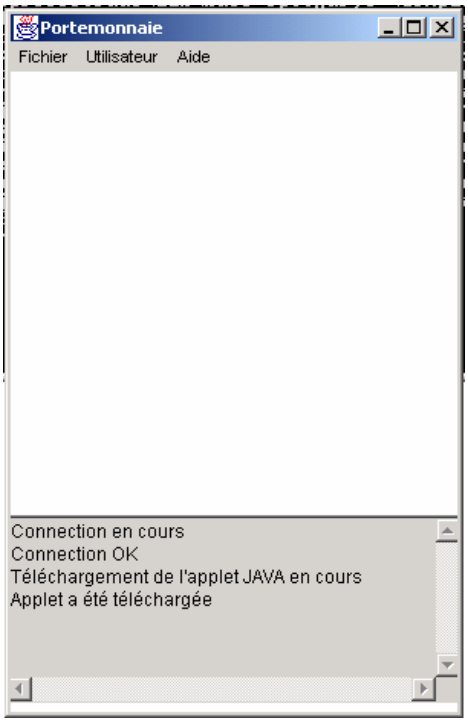

Une fois que le lecteur est connecté et avant de télécharger l'applet dans la carte, on lance le Jcard Manager et sélectionne de la liste des commandes, la commande **Get Memory Space** qui nous permet de savoir l'espacement mémoire disponible dans la carte**.**



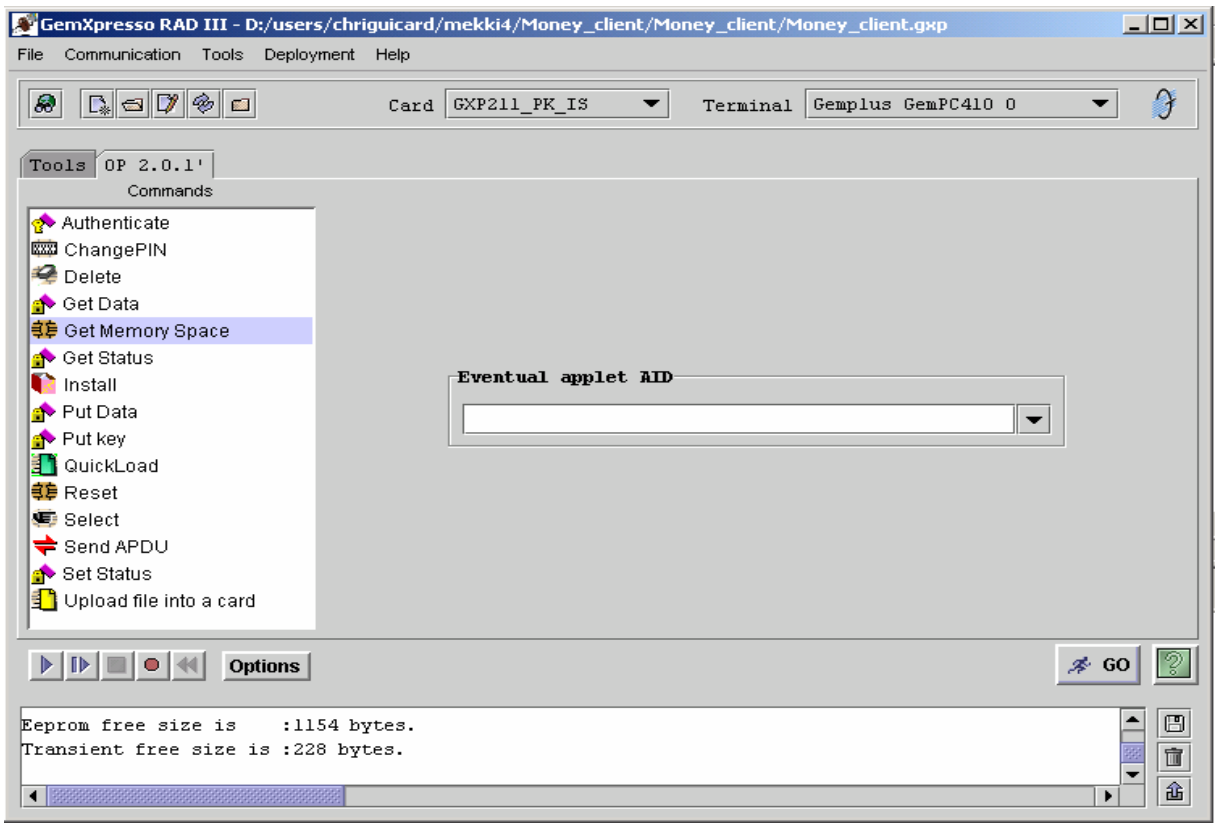

En cliquant le bouton **GO,** l'espace mémoire libre disponible s'affiche dans la fenêtre d'affichage de la Jcard Manager indiquant dans notre exemple un espace de 17728 octets.

Si l'applet n'a pas été déjà téléchargé dans, on effectuera cette opération de la manière suivante :



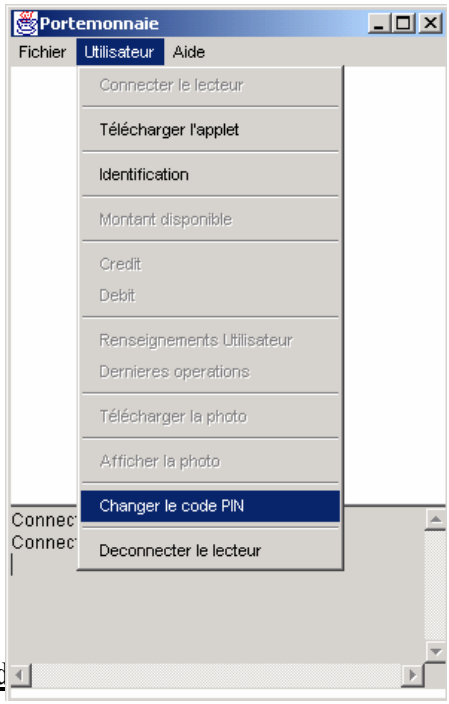




En téléchargeant l'applet, l'espace mémoire va bien évidemment diminuer et l'applet va occuper un espace de 2 Koctets presque.



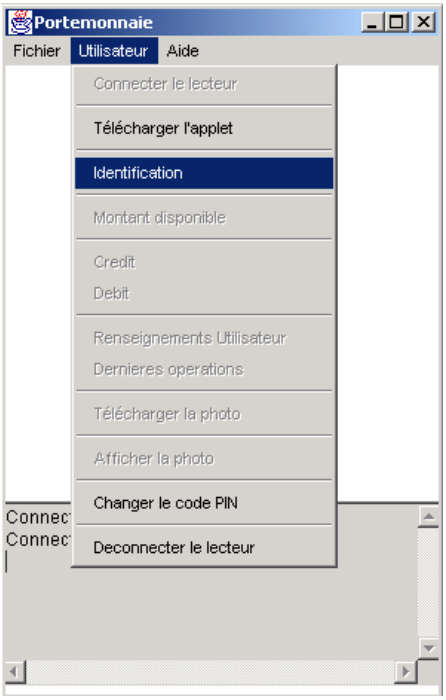
Pour réaliser toutes les fonctions d'un porte-monnaie et visualiser une photographie, on doit bien s'identifier à l'aide d'un code d'accès. Par ailleurs, il est possible de changer le code PIN.

**Changement du code PIN :**

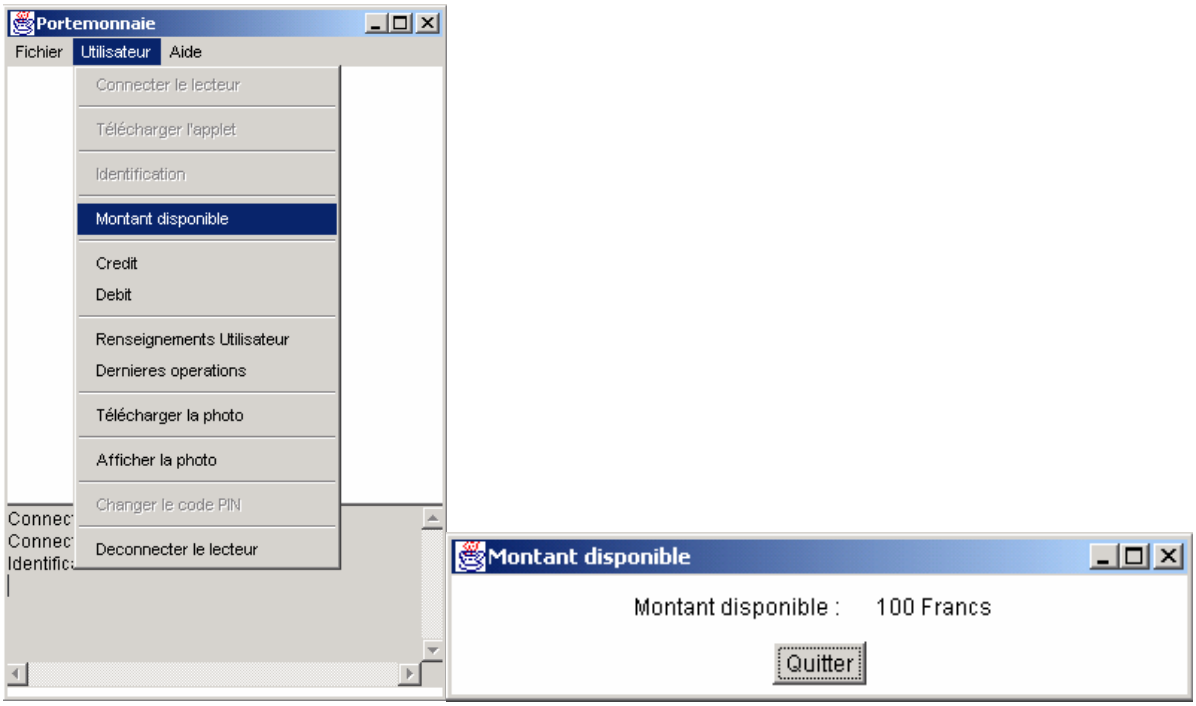




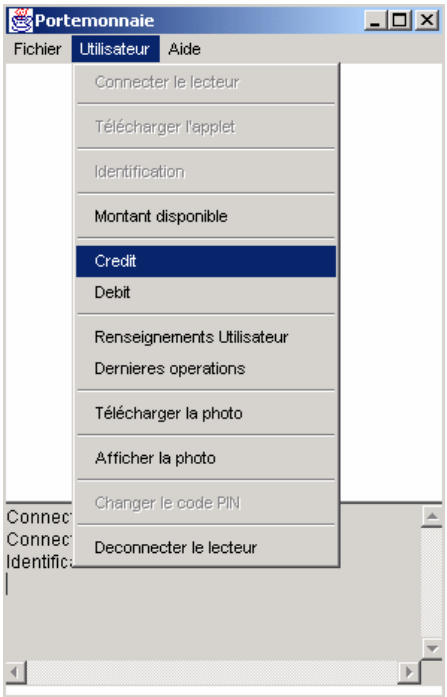
**Id**

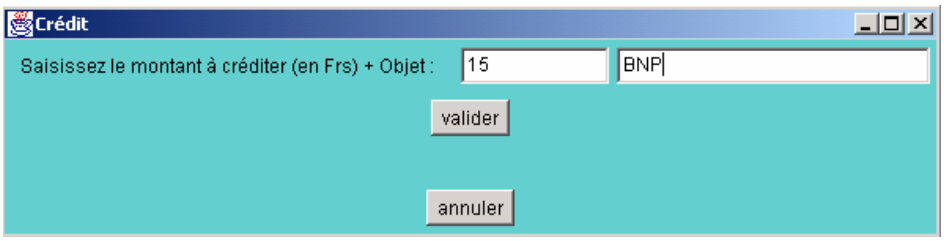



Après avoir bien s'identifier pour avoir accès aux informations stockées dans la carte, on à présent visualiser le montant disponible par exemple.



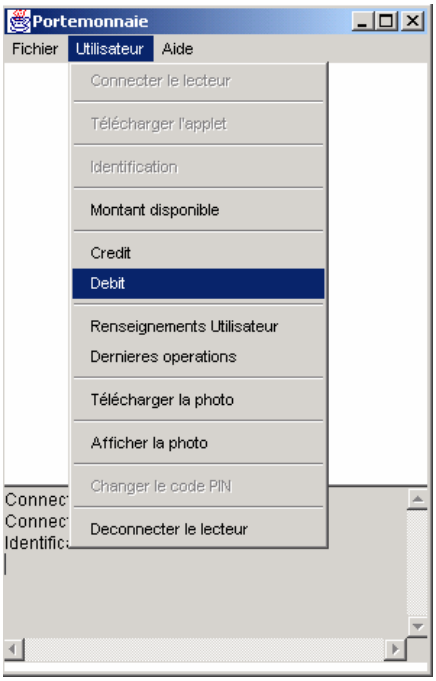

Ensuite, on peut réaliser un crédit tout en saisissant le montant et l'objet.

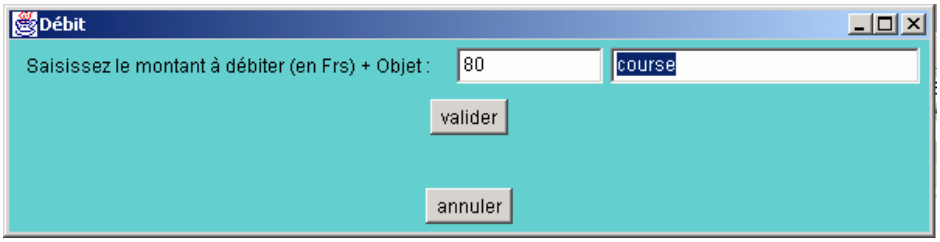





Le nouveau montant est affiché automatiquement :

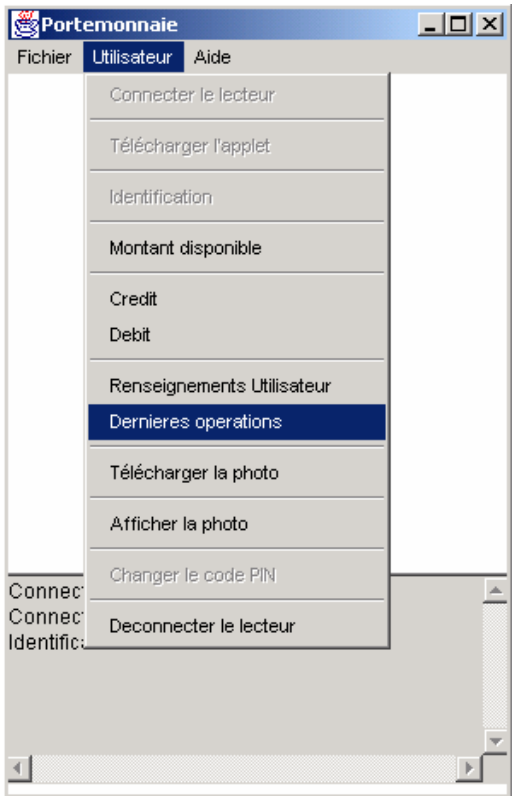

A présent, on effectue un débit en indiquant le montant et l'objet :



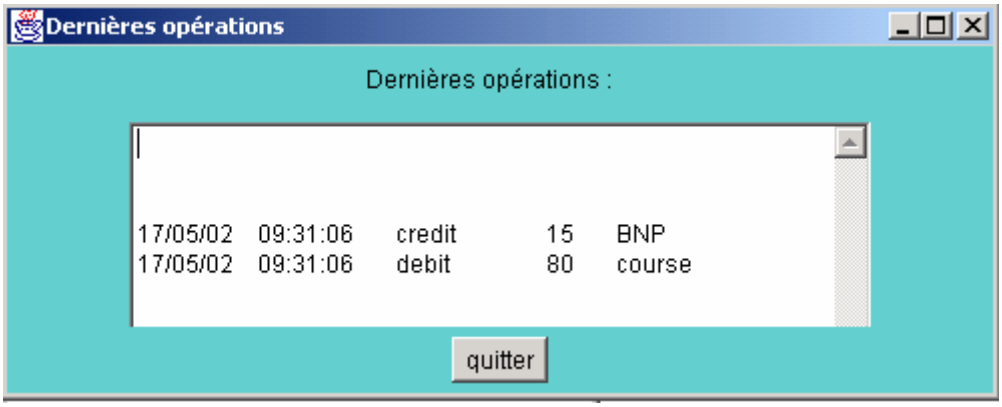




Le nouveau montant est affiché automatiquement:

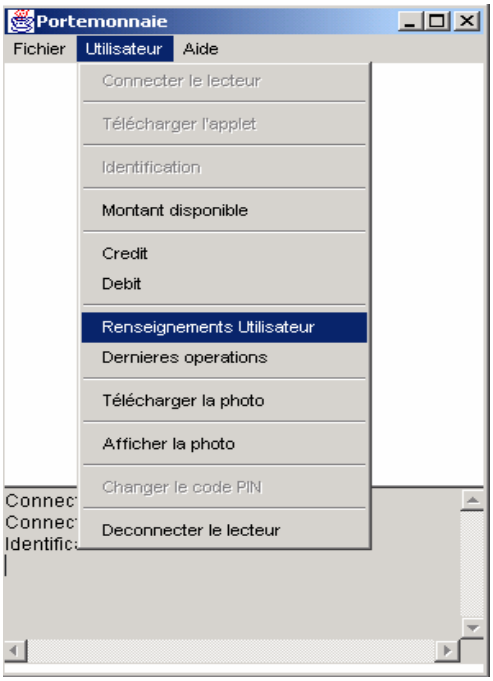

Une fonction nous permet d'afficher les dernières opérations exécutées:

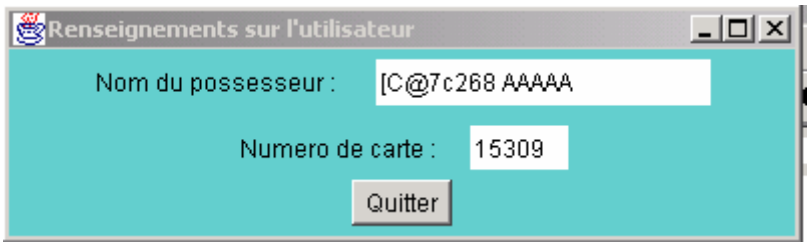



On observe différents renseignements sur les opérations effectuées comme la date, l'heure, l'objet et les montants:



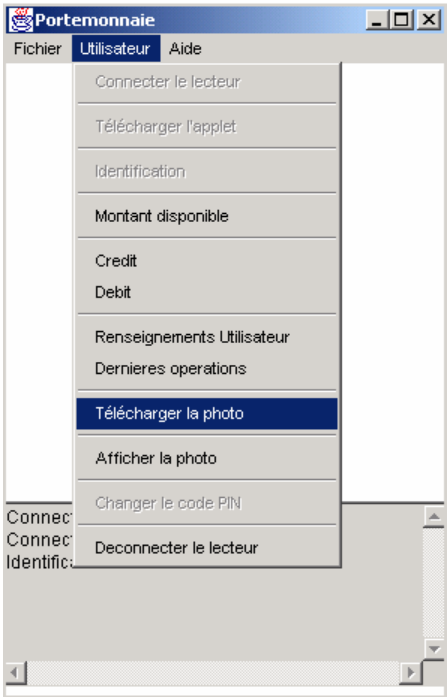

Il existe une autre fonction qui peut renseigner l'utilisateur:

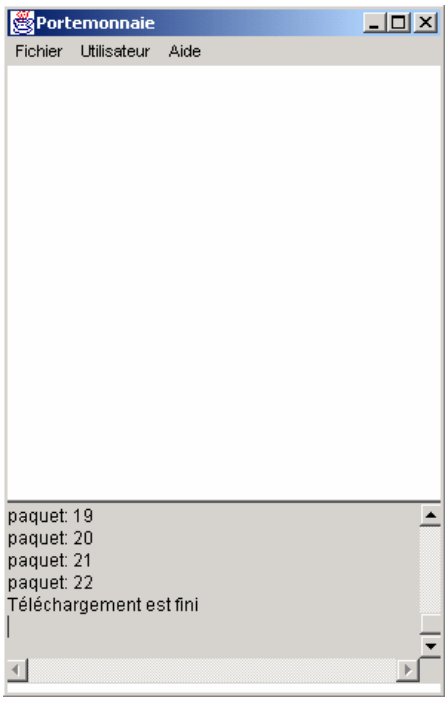


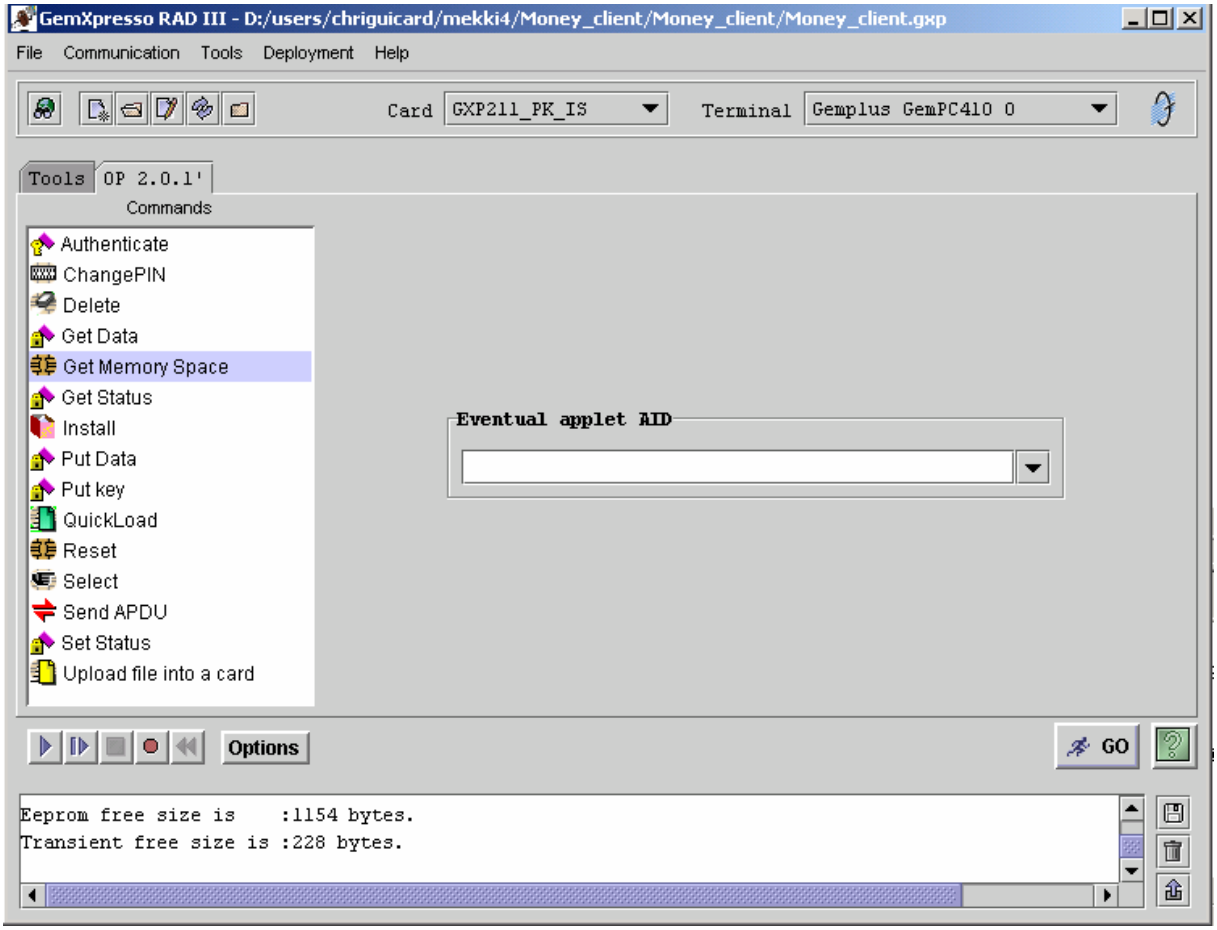

Maintenant, on va visualiser une photo quelconque. Pour réaliser cette tâche, on doit tout d'abord télécharger la photo dans la carte pour pouvoir l'afficher par la suite.

**Téléchargement de la photo:**

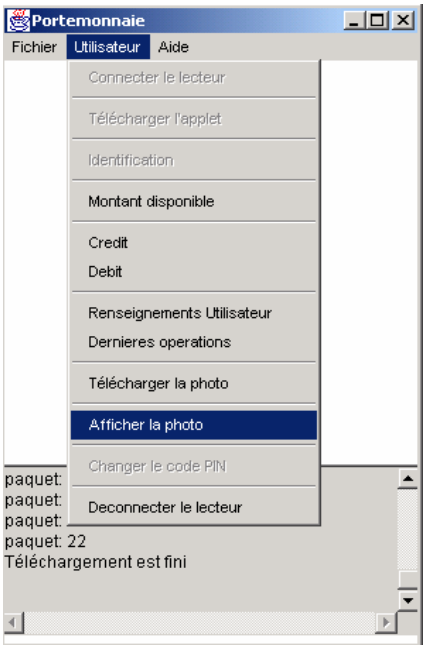


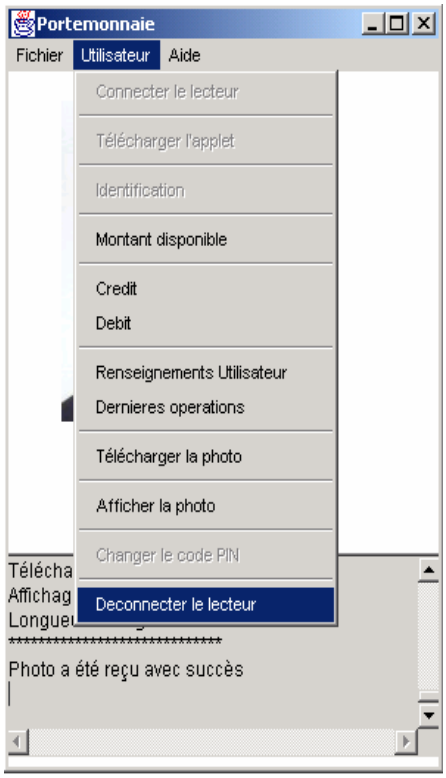
Après avoir télécharger l'applet qu'a occupé presque 2 Koctets, il reste un espace au de 15 kilos. Ceci veut dire qu'on n'a pas le droit au téléchargement d'une photo ayant une taille dépassant 15 kilos d'octet.




## Affichage de la photo:




Finalement, on peut déconnecter le lecteur de la façon suivante:




# Conclusion

## Les avantages des cartes.

Les circuits électroniques intégrés mettent à notre disposition dans un même objet les trois facteurs fondamentaux suivants que sont **: la puissance de traitement ou de calcul, la capacité de mémorisation** ainsi que celle **de communication**. Le succès de la carte à puce dans la société de l'information repose sur **sa capacité à sécuriser les systèmes** dans lesquels elle s'insère. Ses caractéristiques particulières, en particulier **sa portabilité** lui confèrent des avantages très recherchés dans la société à laquelle nous accédons. On peut tenter d'en dresser un bref panorama :

**A) Extrême difficulté de contrefaçon** : Différentes méthodes sophistiquées sont employées pour dissuader la copie et détecter les attaques. Une panoplie de détecteurs et capteurs divers et de dispositifs de protection sont intégrés dans la carte.

**B) Fiabilité de l'identification** : L'addition de la possession d'un code secret et de la carte renforce considérablement la sécurité de l'identification.

**C) La sécurité ultime sera fournie par l'utilisation combinée de la carte à puce et de la biométrie**. Des solutions économiques vont peu à peu émerger pour éviter aux porteurs d'avoir à se souvenir de mots de passe. La reconnaissance de la voix, des empreintes digitales, de la rétine sont des pistes explorées activement. Dans tous ces cas, une référence biométrique peut être enregistrée dans la carte.

En définitive, la carte à puce constitue la synergie idéale réunie au sein du même objet nomade sécurisé entre l'électronique et la cryptographie pour protéger les informations personnelles qui s'avèrent indispensables dans la vie sociale moderne.

# Bibliographie


- Di Giorgio, R. S. Smart Cards: A primer. Java World2, 12 (Dec. 1997). [http://www.javaworld.com/javaworld/jw-12-1997/jw-12-javadev.html].
- Di Giorgio, R. S. understanding java card 2.0. java World 3, 3( Mar. 1998) [http://www.javaworld.com/javaworld/jw-03-1998/jw-03-javadev.html].
- GEMPLUS. GemCore-Based Reader Interface Library Programmer's Guide, May 1997.
- LANET, J. –L., AND REQUET, A. Format Proof of Smart Card Applets Correctness. In Quisquater and Schneier. [http://www.gemplus.fr/developers/passwd-protected/trends/publications/formproof/-art3.htm]
- SUN MICROSYSTEMS, INC. Java Card 2.0 Language Subset and Virtual Machine Specification, Programming Concepts, and Application Programming Interfaces, Oct. 1997. [http://java.sun.com/products/javacard/].
- SUN MICROSYSTEMS, INC. Java Card 2.1 Virtual Machine, Runtime Environment, and Application Programming Interface Specifications, Public Review ed , Feb 1999. [http://java.sun.com/products/javacard/javacard21.html].
- SUN MICROSYSTEMS, INC. Java Card Runtime Environment (JCRE) 2.1Specification, Final Revision 1.0 ed , Feb. 1999. [http://java.sun.com/products/javacard/JCRESpec.pdf].
- Vandewalle, J.-J., AND Vétillard, E. Developing Smart Card-Based Application using Java Card. In 3rd Smart Card Research and Advanced Applications Conference [33]. [http://www.gemplus.fr/developers/passwd-protected/trends/publications/devscappli/-art1.htm].