# ED3 sur la Technologie Java Card

**Samia Bouzefrane & Julien Cordry**

**Laboratoire CEDRIC**

**CNAM**

# Comment écrire une applet ?

# Exemple du porte-monnaie électronique

**1. Comment écrire une applet ?**

Quatre étapes comprennent la phase de développement d'une applet:

1. Spécifier les fonctions de l'applet
2. Assigner des AIDs à l'applet et au paquetage contenant la classe de l'applet
3. Concevoir les programmes de l'applet
4. Définir l'interface entre l'applet et le terminal

Nous illustrons ces étapes à travers l'exemple du porte-monnaie.

**2. Spécification des fonctions de l'applet :**

Notre applet porte-monnaie va stocker de la monnaie électronique et supporte les fonctions de crédit, débit et de contrôle de la balance. Pour éviter l'utilisation frauduleuse de la carte, la carte contient un algorithme de sécurité. Cet algorithme exige à l'utilisateur d'entrer un PIN, une chaîne de 8 chiffres au plus. L'utilisateur de la carte tape son PIN sur un keypad relié au terminal. L'algorithme de sécurité provoque le verrouillage de la carte au bout de 3 tentatives échouées lors de la saisie du PIN. Le PIN est initialisé lors de l'installation de l'applet.

Le PIN doit être vérifié avant l'exécution de toute transaction de crédit ou de débit.

Pour simplifier, supposons que le maximum de la balance est de 32 767€ (15 bits + 1 bit de signe) et que toute transaction de débit ou de crédit ne peut dépasser 127€ (7 bits + 1 bit de signe). Ainsi, les variables Java de type short et byte peuvent représenter respectivement la balance et le montant de chaque transaction.

**3. Spécification des AIDs :**

Dans la technologie Java Card, chaque applet est identifiée et sélectionnée par un identificateur (AID). De même, à chaque paquetage Java est assigné un AID. Cette convention de nommage est conforme à la spécification de la carte à puce définie dans l'ISO 7816.

Un AID est une séquence d'octets allant de 5 à 16 octets. Son format est donné au Tableau 1.

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

**Tableau 1. Format de l'AID**

L'ISO gère l'affectation des RIDs aux compagnies, chaque compagnie obtient son propre et unique RID de l'ISO. Les compagnies gèrent l'affectation des PIXs pour leurs AIDs.

Les classes Java de l'applet sont définies dans un paquetage Java. Leurs AIDs respectifs sont définies dans le Tableau 2.

| **Package AID** | | |
|---|---|---|
| **Champ** | **Valeur** | **Longueur** |
| RID | 0xA0, 0x00, 0x00, 0x18, 0x50 | 5 octets |
| PIX | 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x52, 0x41, 0x44, 0x50 | 10 octets |
| **Applet AID** | | |
| **Champ** | **Valeur** | **Longueur** |
| RID | 0xF2, 0x34, 0x12, 0x34, 0x56 | 5 octets |
| PIX | 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x52, 0x41, 0x44, 0x41 | 10 octets |

**Tableau 2. AIDs fictifs pour l'applet et son paquetage**

**4. Définir les méthodes de l'applet :**

Une applet Java Card doit étendre la classe `javacard.framework.Applet`. Cette classe est une superclasse des applets résident sur la carte. Elle définit les méthodes courantes que doit utiliser une applet pour interagir avec le JCRE, l'environnement d'exécution.

Tableau 3 liste les méthodes de type `public` et `protected` définies dans la classe `javacard.framework.Applet`:

| **Method summary** | |
|---|---|
| `public void` | `deselect ()`<br><br>Called by the JCRE to inform the currently selected applet that another (or the same) applet will be selected. |
| `public Shareable` | `getShareableInterfaceObject (AID client AID, byte parameter)`<br><br>Called by the JCRE to obtain a sharable interface object from this server applet on behalf of a request from a client applet. |
| `public static void` | `install (byte[] bArray, short bOffset, byte bLength)`<br><br>The JCRE calls this static method to create an instance of the `Applet` subclass. |

| | |
|---|---|
| `public abstract void` | `process (APDU apdu)`<br><br>Called by the JCRE to process an incoming APDU command. |
| `protected final void` | `register ()`<br><br>This method is used by the applet to register this applet instance with the JCRE and assign the default AID in the CAD file to the applet instance. |
| `protected final void` | `register (byte[] bArray, short bOffset, byte bLength)`<br><br>This method is used by the applet to register this applet instance with the JCRE and to assign the specified AID in the array `bArray` to the applet instance. |
| `public boolean` | `select ()`<br><br>Called by the JCRE to inform this applet that it has been selected. |
| `protected final boolean` | `selectingApplet ()`<br><br>This method is used by the applet `process()` method to distinguish the `SELECT APDU` command that selected this applet from all other `SELECT APDU` APDU commands that may relate to file or internal applet state selection. |

**Tableau 3. Méthodes `public` et `protected` définies dans la classe `javacard.framework.Applet`**

La classe `javacard.framework.Applet` fournit un framework pour l'exécution des applets. Les méthodes définies dans la classe sont appelées par le JCRE lorsque celui-ci reçoit des commandes APDU à partir du lecteur (CAD :Card Acceptance Device).

Une fois le code de l'applet proprement chargé sur la carte et lié aux autres paquetages se trouvant sur la carte, la vie de l'applet commence lorsqu'une instance de l'applet est créée et enregistrée au sein du JCRE. Une applet doit implémenter la méthode `install`() pour créer une instance d'applet et doit enregistrer l'instance au sein du JCRE en invoquant une des méthodes `register`(). La méthode `install`() prend un vecteur d'octets comme paramètre. Ce vecteur contient les paramètres d'installation pour l'initialisation et la personnalisation de l'instance d'applet.

Une applet Java Card reste inactive jusqu'à ce qu'elle soit explicitement sélectionnée. Lorsque le JCRE reçoit une commande `SELECT APDU`, il consulte sa table interne pour trouver l'applet dont l'AID correspond à celui spécifié dans la commande. S'il le trouve, le JCRE prépare la sélection de la nouvelle applet. Cette préparation se fait en deux étapes: d'abord, si une applet couramment sélectionnée est présente en mémoire, alors le JCRE la désélectionne en invoquant la méthode `deselect`(). L'applet exécute la méthode `deselect`() avant de devenir inactive. Le JCRE invoque alors la méthode `select`() pour informer la nouvelle applet de sa sélection. La nouvelle applet effectue éventuellement une opération d'initialisation avant d'être réellement sélectionnée. L'applet retourne vrai à la méthode `select`() si elle est prête à devenir active et à traiter n'importe quelle commande APDU. Sinon, l'applet

retourne faux pour décliner sa participation. La classe `javacard.framework.Applet` fournit une implémentation par défaut pour les méthodes `select`() et `deselect`(). Une sous-classe de la classe `Applet` peut redéfinir ces méthodes pour associer un autre comportement à ces méthodes.

Une fois l'applet sélectionnée, le JCRE fait suivre toutes les commandes APDU (y compris la commande `SELECT`) à la méthode `process`() de l'applet. Dans la méthode `process`(), l'applet interprète chaque commande APDU et exécute la tâche spécifiée par la commande. Pour chaque commande APDU, l'applet répond au CAD en envoyant une réponse APDU qui informe le CAD du résultat du traitement de la commande APDU. La méthode `process`() de la classe `javacard.framework.Applet` est une méthode de type `abstract`: une sous-classe de la classe `Applet` doit redéfinir cette méthode pour implémenter les fonctions de l'applet.

Ce dialogue commande-réponse continue jusqu'à ce que une nouvelle applet soit sélectionnée ou bien que la carte soit retirée du CAD. Lorsqu'elle est désélectionnée, l'applet devient inactive jusqu'à sa prochaine sélection.

La méthode `getShareableInterfaceObject` sert dans la communication inter-applet. Elle est appelée par une applet client qui demande, à une applet serveur, à partager l'interface d'un objet. L'implémentation par défaut de cette méthode est de retourner null. Dans ce document, on ne s'intéressera pas à la communication entre applets.

Étant donné que la commande APDU `SELECT` est aussi dirigée vers la méthode `process`(), la méthode `selectingApplet`() est utilisée par la méthode `process`() de l'applet pour distinguer entre la commande APDU `SELECT` qui sélectionne cette applet et les autres commandes APDU `SELECT` relatives à la sélection de fichiers ou de l'état interne de l'applet.

**5. Définir une interface entre une applet et le terminal:**

Une applet qui tourne sur une carte à puce communique avec l'application en utilisant le protocole APDU (Application Protocol Data Units défini par l'ISO 7816). Par essence, l'interface entre l'applet et l'application est un ensemble de commandes APDU qui sont supportées aussi bien par l'applet que l'application.

**5.1 La notion d'APDU:**

Les commandes APDU sont toujours des ensembles de paires. Chaque paire contient une commande (*command*) *APDU,* qui spécifie une commande, et une réponse (*response) APDU,* qui retourne le résultat d'exécution de la commande. Dans le monde de la carte, les cartes sont *réactives* – c-à-d qu'elles n'initient jamais des

communications mais se contentent de répondre aux APDUs du monde extérieur. L'application envoie une commande APDU via le lecteur (CAD). Le JCRE en recevant une commande, sélectionne une nouvelle applet ou bien passe la commande à une applet courante (déjà sélectionnée). L'applet courante traite la commande et retourne une réponse APDU à l'application. Les commandes et réponses APDU sont échangées alternativement par la carte et le CAD.

Tableau 4 décrit le format des commandes et réponses APDU.

<table>
<tr><th colspan="7">Command APDU</th></tr>
<tr><th colspan="5">Mandatory header</th><th colspan="2">Optional body</th></tr>
<tr><td>CLA</td><td>INS</td><td>P1</td><td>P2</td><td>Lc</td><td>Data field</td><td>Le</td></tr>
<tr><td colspan="7"><ul><li>CLA (1 byte): Class of instruction --- indicates the structure and format for a category of command and response APDUs</li><li>INS (1 byte): Instruction code: specifies the instruction of the command</li><li>P1 (1 byte) and P2 (1 byte): Instruction parameters -- further provide qualifications to the instruction</li><li>Lc (1 byte): Number of bytes present in the data field of the command</li><li>Data field (bytes equal to the value of Lc): A sequence of bytes in the data field of the command</li><li>Le (1 byte): Maximum of bytes expected in the data field of the response to the command</li></ul></td></tr>
<tr><th colspan="7">Response APDU</th></tr>
<tr><th colspan="4">Optional body</th><th colspan="3">Mandatory trailer</th></tr>
<tr><td colspan="4">Data field</td><td>SW1</td><td colspan="2">SW2</td></tr>
<tr><td colspan="7"><ul><li>Data field (variable length): A sequence of bytes received in the data field of the response</li><li>SW1 (1 byte) and SW2 (1 byte): Status words -- denote the processing state in the card</li></ul></td></tr>
</table>

**Tableau 4. formats des commandes et réponses APDU**

### 5.2 Définir des commandes APDU:

Une applet Java Card doit supporter un ensemble de commandes APDU, comprenant la commande `SELECT APDU` et une ou plusieurs commandes de traitement d'APDUs.

La commande `SELECT` invite le JCRE à sélectionner une applet sur la carte.

L'ensemble des commandes de traitement (process) définit les commandes que l'applet supporte. Elles sont définies en accord avec les fonctions de l'applet.

La technologie Java Card spécifie l'encodage de la commande `SELECT APDU`. Les développeurs sont libres de définir l'encodage des commandes de traitement. Cependant, les commandes de traitement doivent être cohérentes avec la structure définie au Tableau 4.

D'un point de vue structurel, la commande `SELECT` et les commandes de traitement sont des paires de commandes et de réponses APDU.

Pour chaque commande APDU, l'applet doit d'abord décoder la valeur de chaque champ de la commande. Si les champs optionnels sont inclus, l'applet doit aussi déterminer leur format et leur structure. En utilisant ces définitions, l'applet sait comment interpréter chaque commande et lire chaque donnée. Elle peut alors exécuter la tâche spécifiée par la commande.

Pour chaque réponse APDU, l'applet doit définir un ensemble de mots d'état pour indiquer le résultat du traitement de la commande APDU. Dans un traitement normal, l'applet retourne un mot d'état contenant succès (0x9000, comme spécifié dans l'ISO 7816). Si une erreur survient, l'applet doit retourner un mot d'état différent de 0x9000 pour exprimer son état interne. Si par contre le champ de donnée optionnel est inclus dans la réponse APDU, l'applet doit définir ce qu'elle doit retourner.

Dans notre exemple, l'applet supporte les fonctions `credit`, `debit`, et `check-balance`. De plus, elle doit supporter la commande `VERIFY` pour vérifier le PIN.

La commande `SELECT` ainsi que les 4 fonctions de traitement d'APDU sont illustrées dans le Tableau 5.

| **Commande APDU VERIFY** | | | | | | |
|---|---|---|---|---|---|---|
| **Commande APDU** | | | | | | |
| **CLA** | **INS** | **P1** | **P2** | **Lc** | **Data field** | **Le** |
| 0xB0 | 0x20 | 0x0 | 0x0 | Longueur du PIN | Valeur du PIN | N/A |
| • Le code INS indique une instruction de vérification avec des paramètres nuls<br>• Le champ données contient la valeur du PIN | | | | | | |
| **Réponse APDU** | | | | | | |
| **Optional data** | **Status word** | **Meaning of status word** | | | | |
| Pas de données | 0x9000 | Traitement avec succès | | | | |
| | 0x6300 | Echec de la vérification | | | | |
| **Commande APDU CREDIT** | | | | | | |
| **Commande APDU** | | | | | | |
| **CLA** | **INS** | **P1** | **P2** | **Lc** | **Data field** | **Le** |
| 0xB0 | 0x30 | 0x0 | 0x0 | 1 | Valeur à créditer | N/A |

- La champ données contient la valeur à créditer

| Réponse APDU | | |
|---|---|---|
| **Optional data** | **Status word** | **Meaning of status word** |
| N/A | 0x9000 | Traitement avec succès |
| | 0x6301 | Exige la vérification du PIN |
| | 0x6A83 | Valeur à créditer invalide |
| | 0x6A84 | Dépasse la valeur maximale autorisée |

**Commande APDU DEBIT**

| Commande APDU | | | | | | |
|---|---|---|---|---|---|---|
| **CLA** | **INS** | **P1** | **P2** | **Lc** | **Data field** | **Le** |
| 0xB0 | 0x40 | 0x0 | 0x0 | 1 | Valeur à débiter | N/A |

- Le champ données contient la valeur à débiter

| Réponse APDU | | |
|---|---|---|
| **Optional data** | **Status word** | **Meaning of status word** |
| N/A | 0x9000 | Traitement avec succès |
| | 0x6301 | Nécessite la vérification du PIN |
| | 0x6A83 | Valeur à débiter invalide |
| | 0x6A85 | Balance négative |

**Commande APDU GET BALANCE**

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

- Le champ données contient la valeur courante de la balance

| Response APDU | | |
|---|---|---|
| **Optional data** | **Status word** | **Meaning of status word** |
| N/A | 0x9000 | Traitement avec succès |

**Tableau 5. Les commandes APDU de l'applet porte-monnaie**

En plus des mots d'état déclarés dans chaque commande APDU, l'interface `javacard.framework.ISO7816` définit un ensemble de mots d'état qui signalent les erreurs courantes des applets, comme par exemple la commande APDU concernant l'erreur du formatage.

**6. Le support APDU dans la technologie Java Card:**

La classe `javacard.framework.APDU` encapsule les commandes APDU. Elle fournit une interface puissante et flexible qui permet aux applets de gérer les commandes APDU. La classe APDU est conçue pour cacher les complexités du protocole, afin que les développeurs d'applet se concentrent davantage sur les détails de l'application.

Lorsque le JCRE reçoit une commande APDU, il encapsule la commande dans un objet APDU qu'il passe à la méthode `process`() de l'applet courante. L'objet APDU comporte un vecteur d'octets qui contient le message APDU.

L'applet traite une commande APDU en invoquant des méthodes sur cet objet APDU. En général, l'applet effectue les étapes suivantes:

**Étape 1. Extraire le buffer APDU:**

L'applet invoque la méthode `getBuffer`() afin d'obtenir une référence au buffer APDU, qui contient le message. Lorsque l'applet reçoit l'objet APDU, seuls les 5 premiers octets sont disponibles dans le buffer. Il s'agit dans l'ordre des octets `CLA`, `INS`, `P1`, `P2`, et `P3`. L'octet `P3` désigne l'octet `Lc`, si la commande possède des données optionnelles. L'applet peut vérifier les octets entête pour déterminer la structure de la commande et l'instruction spécifiée par la commande.

**Étape 2. Recevoir des données:**

Si la commande APDU contient des données optionnelles, l'applet doit diriger l'objet APDU vers la réception de données entrantes en invoquant la méthode `setIncomingAndReceive`(). Les données sont lues dans le buffer APDU en suivant l'ordre des 5 octets d'entête. Le dernier octet de l'entête (`Lc`) indique la longueur des données entrantes. Si le buffer APDU ne peut contenir toutes les données, l'applet peut traiter les données en fragments, ou bien le copier vers un buffer interne. Dans les deux cas, elle doit faire appel de manière répétitive à la méthode `receiveBytes`() afin de lire les données additionnelles à partir du buffer APDU.

**Étape 3. Renvoyer des données:**

Après avoir traité une commande APDU, l'applet peut retourner des données à l'application sous forme de réponses APDU. L'applet doit d'abord faire appel à la

méthode `setOutgoing`() pour obtenir la longueur de la réponse (`Le`). `Le` est spécifié dans la commande APDU associée à la réponse APDU.

Ensuite, l'applet appelle la méthode `setOutgoingLength`()pour informer le CAD de la longueur réelle des données de la réponse. L'applet peut transférer les données vers le buffer APDU et appeler la méthode `sendBytes`() pour envoyer les données. La méthode `sendBytes`() peut être appelée plusieurs fois si le buffer APDU ne peut pas contenir toutes les données retournées.

Si les données sont stockées dans un buffer interne, l'applet invoque la méthode `sendByteLong`() pour envoyer les données à partir du buffer.

Si les données de la réponse sont trop courtes pour tenir dans le buffer APDU, la classe APDU fournit une méthode appropriée: `setOutgoingAndSend`(). Cette méthode est une combinaison de `setOutgoing`, de `setOutgoingLength` et de `sendBytes`. Néanmoins cette méthode ne peut être invoquée qu'une seule fois, et aucune méthode d'envoi ne peut être appelée après.

**Étape 4. Renvoyer le mot d'état (word status):**

Après un succès de la méthode `process`(), le JCRE envoie automatiquement 0x9000 pour indiquer un traitement normal. A n'importe quel point, si l'applet détecte une erreur, l'applet peut lever l'exception `ISOException` en appelant la méthode statique `ISOException.throwIt(short reason)`. Le mot d'état est spécifié dans le paramètre `reason`. Si l'exception `ISOException` n'est pas gérée par l'applet, elle sera attrapée par le JCRE. Le JCRE extrait le code de `reason` code et l'envoie comme mot d'état.

**7. Cas particulier du PIN**

Le package javacard.framework fournit une interface PIN pour la manipulation du PIN. La classe OwnerPIN (qui implémente l'interface PIN) représente la PIN propriétaire. Elle implante la fonctionnalité "Personal Identification Number" et fournit la possibilité de mettre à jour le PIN.

| Constructor Summary |
|---|
| `OwnerPIN(byte tryLimit, byte maxPINSize)`<br>Constructor. |

Les méthodes de la classe OwnerPIN sont les suivantes :

| Method Summary | |
|---|---|
| `boolean` | `check(byte[] pin, short offset, byte length)` |

| | |
|---|---|
| | Compares `pin` against the PIN value. |
| `byte` | **`getTriesRemaining`**`()`<br>Returns the number of times remaining that an incorrect PIN can be presented before the `PIN` is blocked. |
| `protected boolean` | **`getValidatedFlag`**`()`<br>This protected method returns the validated flag. |
| `boolean` | **`isValidated`**`()`<br>Returns `true` if a valid PIN has been presented since the last card reset or last call to `reset()`. |
| `void` | **`reset`**`()`<br>If the validated flag is set, this method resets it. |
| `void` | **`resetAndUnblock`**`()`<br>This method resets the validated flag and resets the `PIN` try counter to the value of the `PIN` try limit. |
| `protected void` | **`setValidatedFlag`**`(boolean value)`<br>This protected method sets the value of the validated flag. |
| `void` | **`update`**`(byte[] pin, short offset, byte length)`<br>This method sets a new value for the PIN and resets the `PIN` try counter to the value of the `PIN` try limit. |

### 8. Construire le code de l'applet:

Une fois la phase de conception de l'applet est finie, la seconde phase consiste à écrire le code de l'applet.

| | |
|---|---|
| `package cnam;` | Package |
| `import javacard.framework.*;` | |
| `public class APurse extends Applet {` | Une applet est une instance d'une classe qui hérite de : `javacard.framework.Applet.` |
| `/* declaration de constantes */`<br>`// code CLA pour`<br>`final static byte APP_CLA = (byte)0xB0;` | CLA identifie une famille de commandes. |
| `// codes INS des commandes APDU`<br>`final static byte VERIF_INS = (byte)0x20;`<br>`final static byte CREDIT_INS = (byte)0x30;`<br>`final static byte DEBIT_INS = (byte)0x40;`<br>`final static byte GET_BALANCE_INS = (byte)0x50;` | INS définit le type d'instruction |
| `// Poids le plus faible a droite`<br>`final static short MAX_BALANCE =` | Déclarer la balance maximale et la valeur maximale de la transaction. |

| | |
|---|---|
| (short)0x7fff;<br> final static byte<br>MAX_TRANSACTION_AMOUNT=127; | |
| // nb de tentatives<br>final static byte PIN_NB_LIMIT =<br>(byte)0x03;<br>// taille maximale du PIN<br>final static byte<br>MAX_PIN_SIZE=(byte)0x08; | Les paramètres du PIN |
| // échec de la vérification du PIN<br>  final static short<br>SW_VERIFICATION_FAILED = 0x6300;<br><br>// indique que la transaction debit ou<br>credit<br>// necessite la verification du PIN<br>  final static short<br>SW_PIN_VERIFICATION_REQUIRED = 0x6301;<br><br>// transaction invalide car<br>// montant>MAX_TRANSACTION_AMOUNT ou<br>montant<0<br>  final static short<br>SW_INVALID_TRANSACTION_AMOUNT = 0x6A83;<br><br>// indique que la balance dépasse le<br>maximum<br>final static short<br>SW_EXCEED_MAXIMUM_BALANCE = 0x6A84;<br><br>//indique que la balance est devenue<br>négative<br>final static short SW_NEGATIVE_BALANCE<br>= 0x6A85; | Les Status Words |
| /* declaration de variables<br>d'instances */<br>  OwnerPIN **pin**;<br>  short **balance**; | |
| private **APurse** (byte[] bArray, short<br>bOffset, byte bLength){<br><br>//crée un objet qui est le PIN de la<br>carte<br>pin = new<br>**OwnerPIN**(PIN_NB_LIMIT,MAX_PIN_SIZE);<br><br>// les paramètres d'installation<br>comprennent<br>// l'initialisation de la valeur du<br>PIN<br>pin.**update**(bArray, bOffset, bLength);<br>**register**();<br>  } // fin du constructeur | Constructeur privé de la classe APurse<br>- alloue de la mémoire aux objets créés<br>- enregistrement de l'applet auprès du JCRE à l'aide de son AID<br><br>La méthode update() affecte une nouvelle valeur au PIN et initialise le nb de tentatives de PIN avec le nb limite PIN_NB_LIMIT. |
| public static void **install**(byte[]<br>bArray, short bOffset, byte bLength) { | Méthode install invoquée par le JCRE pour installer l'applet |

| | |
|---|---|
| `// crée une instance de l'applet APurse`<br>`  new APurse(bArray, bOffset, bLength);`<br>`  } // fin de la méthode install` | |
| `  public boolean select() {`<br>` // l'applet refuse d'etre selectionnee`<br>`// si le PIN est bloque`<br>`if ( pin.getTriesRemaining() == 0 )`<br>`  return false;`<br>`else return true;`<br>`  }// fin de la méthode select` | Méthode appelée par le JCRE pour indiquer la sélection de l'applet |
| `  public void deselect() {`<br>`  // remise à zero de la valeur du PIN`<br>`  pin.reset();`<br>`  }` | Cette méthode est appelée par le JCRE pour effectuer une éventuelle désallocation. |
| `  public void process(APDU apdu) {`<br>` // le buffer qui recupere l'objet APDU`<br>`// ne contient au depart que les octets`<br>`  // [CLA, INS, P1, P2, P3]`<br>`  byte[] buffer = apdu.getBuffer();` | Après sélection d'une apple, le JCRE dirige les commandes APDU vers la méthode process qui les traite. L'object APDU est détenu et maintenu par le JCRE. |
| `  if (selectingApplet()) return;` | Le JCRE passe aussi la commande SELECT APDU à l'applet. |
| `// vérifie le CLA des commandes`<br>`????` | L'exception renvoyée en cas d'erreur est explicitée dans les status word (SW1, SW2). Une exception non capturée par l'applet sera capturée par le JCRE. |
| `// identifier les codes INS`<br>`???`<br>`  } // fin de la methode process` | Les fonctions principales qui réalisent l'action spécifiée par la commande APDU avec un retour d'une réponse APDU. |
| `private void credit (APDU apdu) { ??}` | La méthode `credit` contrôle tous les cas en retournant des exceptions en cas d'erreur. L'objet APDU contient le montant à créditer. |
| `private void debit (APDU apdu) { ???}` | L'objet APDU contient le montant à débiter. |
| `private void getBalance (APDU apdu) {`<br>`???}` | La méthode `getBalance` retourne la balance dans le champ données de la réponse APDU. Etant donné que le champ données est optionnel, l'applet doit informer explicitement |

| | |
|---|---|
| | le JCRE des données additionnelles. |
| `private void verify(APDU apdu) { ???}`<br><br>`} // fin de la classe APurse` | Le PIN est utilise pour protéger la carte à puce. Le nombre de tentatives échouées peut être enregistré. Ainsi la carte sera bloquée si ce nombre dépasse le nombre maximal autorisé. En cas de succès et après sélection de l'applet le PIN doit être vérifié avant toute instruction exécutée par l'applet. |

**Question1**

Utilisez l'environnement Eclipse pour générer votre applet en lui associant un AID. A partir des indications données dans le tableau ci-dessus, écrire une première applet qui n'utilise pas de PIN. Compléter la méthode process() de manière à ce qu'elle puisse reconnaître les différentes instructions et qu'elle puisse lancer les méthodes privées `credit`(), `debit`() ou `getBalance`() en fonction du code instruction lu. Compilez l'applet pour obtenir un fichier .cap. Installer l'applet sur la carte et interagir avec la carte en testant les commandes APDU.

**Question 2**

Reprendre la même applet en intégrant la gestion du PIN.

**Références**

Zhiqun Chen, "**How to write a Java Card applet: A developer's guide**", http://www.javaworld.com/javaworld/jw-07-1999/jw-07-javacard.html.

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

**Global Platform, Card Specification :**
http://www.globalplatform.org/specificationform2.asp?id=archived

**API Java Card** : http://java.sun.com/products/javacard/htmldoc

**ED1 et ED2** : http://cedric.cnam.fr/~bouzefra/javacard.html