

La Carte à Microprocesseur

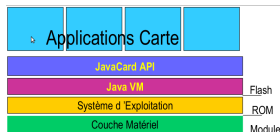
Un système embarqué en plein essor

Tegawendé F. Bissyandé
tegawende.bissyande@fasolabs.org

Cours préparé pour
L'Institut Supérieur de Technologie (IST Burkina)

April 7, 2015

La Java Card



- Pas de chargement dynamique de classe
- Allocation dynamique d'objets supportée mais
 - pas de garbage collection
 - pas de désallocation explicite non plus :(
 - ...
- Quelques types de base (byte, int, boolean) – pas de char (ni de classe String), double, float, long
- Objets supportés / Mécanisme d'héritage supporté / Pas de threads
- Sécurité : Notion de paquetages & modifieurs public, private, protected

Architecture de la Java Card

Applets

loyalty
applet

wallet
applet

authentication
applet

JCRE

framework
classes (APIs)

industry-specific
extensions

installer

system classes

applet
management

transaction
management

I/O network
communication

other
services

Java Card virtual machine
(bytecode interpreter)

native methods

- JCVM : Java Card Virtual Machine

- allocation mémoire
- exécution du bytecode

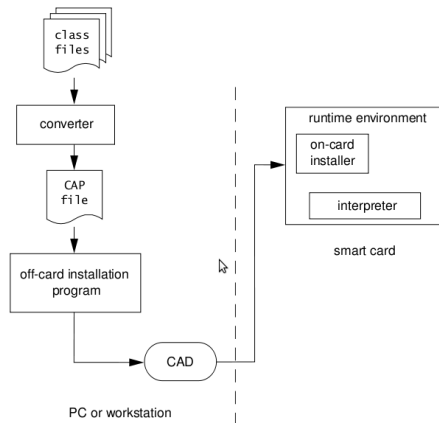
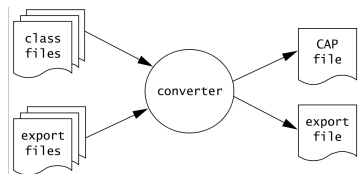
- JCRE : Java card runtime environment

- chargée à l'usine
- copie les données de EEPROM et ROM vers RAM

- native methods

- communication bas niveaux
- support cryptographique

Retour sur la compilation



Construction d'applications Java Card

Une application carte

- Code dans la carte (app serveur = applet Java card)
- Code dans le terminal (app cliente)

Construction d'une application Java card

- Construction de l'applet (implementation des services)
- Installation de l'applet dans les cartes (Initialisation des services)
- Construction de l'application cliente (Invocation des services)

Construction d'applications Java Card

Une application carte

- Code dans la carte (app serveur = applet Java card)
- Code dans le terminal (app cliente)

Construction d'une application Java card

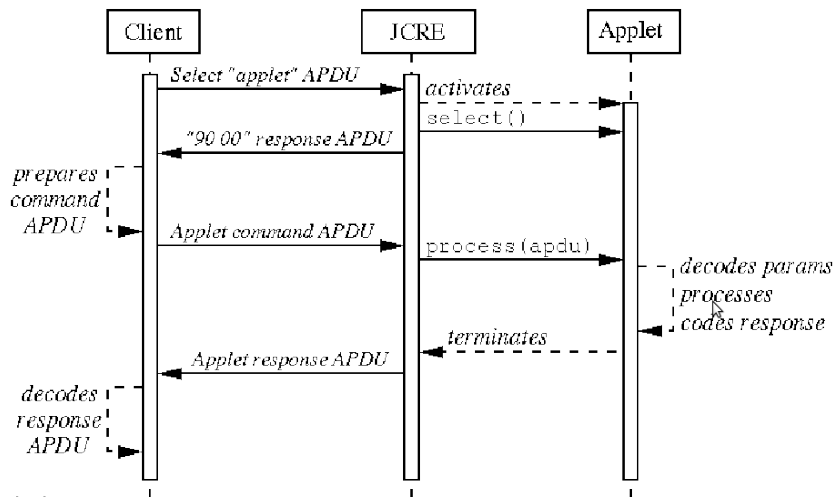
- Construction de l'applet (implementation des services)
- Installation de l'applet dans les cartes (Initialisation des services)
- Construction de l'application cliente (Invocation des services)

- Installation de l'applet Java Card
 - Compilation, conversion et chargement sécurisé de l'applet dans les cartes (Java Card IDE)
 - Appel à la méthode `install(APDU apdu)` des applets (non standardisé)
 - L'APDU contient les paramètres d'initialisation de l'applet

- Construction de l'applet Java Card
 - Implémentation des classes de l'applet avec l'API Java Card
 - Définition des APDUs de commande traités par l'applet et des APDUs de réponse renvoyés par l'applet (données ou erreurs)
 - → implémentation de la méthode `process (APDU apdu)`
 - Le JCRE fournit l'environnement d'exécution et la couche de communication

- Construction de l'application terminal
 - Implémentation des classes du terminal (avec JDK)
 - Communication avec le serveur (applet carte)
 - Établissement de la liaison : envoi d'un APDU de sélection avec l'AID de l'applet (standardisé)
 - Invocation de services de l'applet :
 - codage et envoi d'APDUs de commande conformes à ceux traités par l'applet
 - réception et décodage des APDUs de réponse retournés par l'applet
 - Pas d'API standard de communication avec la carte

Programming.....



Programming....

- Un simple Compteur
 - Carte de fidélité, Porte Monnaie Electronique, ...
- APDUs traités par l'applet :
 - `int lire()`
 - Commande : `AA 01 XX XX 00 04`
 - Réponse : `RV3 RV2 RV1 RV0 90 00`
 - `int incrementer(int)`
 - Commande : `AA 02 XX XX 04 AM3 AM2 AM1 AM0 04`
 - Réponse : `RV3 RV2 RV1 RV0 90 00`
 - `int decrements(int)`
 - Commande : `AA 03 XX XX 04 AM3 AM2 AM1 AM0 04`
 - Réponse : `RV3 RV2 RV1 RV0 90 00`

Programming....

```
package org.carte.compteur ;

import javacard.framework.* ;

public class Compteur extends Applet {
    private int valeur;

    public Compteur() { valeur = 0; register(); }
    public static void install( APDU apdu ) { new Compteur(); }

    public void process( APDU apdu ) {
        byte[] buffer = apdu.getBuffer();
        if ( buffer[ISO.OFFSET_CLA] != 0xAA )
            ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);
        switch ( buffer[ISO.OFFSET_INS] ) {
            case 0x01: ... // Opération de lecture
            case 0x02: ... // Opération d'incrément
            case 0x03: ... // Opération de décrémentation
            default:
                ISOException.throwIt(ISO.SW_INS_NOT_SUPPORTED);
        }
    }
}
```

Programming....

```
case 0x03: // Opération de décrémentation
{
    // Réception des données
    byte octetsLus = apdu.setIncomingAndReceive();
    if ( octetsLus != 4 )
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    int montant = (buffer[ISO.OFFSET_CDATA]<<24) |
        (buffer[ISO.OFFSET_CDATA+1]<<16) |
        (buffer[ISO.OFFSET_CDATA+2]<<8) |
        buffer[ISO.OFFSET_CDATA+3];
    // Traitement
    if ( montant<0 || valeur-montant<0 )
        ISOException.throwIt((short)0x6910);
    valeur = valeur - montant;
    // Envoie de la réponse
    buffer[0] = (byte)(valeur>>24);
    buffer[1] = (byte)(valeur>>16);
    buffer[2] = (byte)(valeur>>8);
    buffer[3] = (byte)(valeur);
    apdu.setOutgoingAndSend((short)0, (short)4);
    return;
}
```

Programming....

```
package com.banque ;

import javacard.framework.*;

public class Pme extends Applet {
    final static byte Pme_CLA      = (byte) 0xB0;
    final static byte Crediter_INS  = (byte) 0x10;
    final static byte Debiter_INS   = (byte) 0x20;
    final static byte Lire_INS      = (byte) 0x30;
    final static byte Valider_INS   = (byte) 0x40;
    final static byte MaxEssai_PIN  = (byte) 0x03;
    final static byte MaxLg_PIN     = (byte) 0x08;
    final static short BalanceNegative_SW = (short) 0x6910;

    OwnerPin pin;
    byte balance;
    byte[] buffer;

    private Pme() {
        pin = new OwnerPIN(MaxEssai_PIN, MaxLg_PIN);
        balance = 0;
        register() ;
    }
}
```

Programming....

```
public static void install(byte[] bArray,
                           short bOffset, byte bLength) {
    Pme p=new Pme();
    pin.updateAndUnblock(bArray, bOffset, bLength);
}

public boolean select() { pin.reset(); return true; }

public void process( APDU apdu ) {
    buffer = apdu.getBuffer();
    if ( buffer[ISO.OFFSET_CLA] != Pme_CLA )
        ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);
    switch ( buffer[ISO.OFFSET_INS] ) {
        case Crediter_INS : crediter(apdu); return;
        case Debiter_INS : debiter(apdu); return;
        case Lire_INS : lire(apdu); return;
        case Valider_INS : valider(apdu); return;
        default:
            ISOEXception.throwIt(ISO.SW_INS_NOT_SUPPORTED);
    }
}
```

Programming....

```
// Réception de données
private void crediter( APDU apdu ) {
    if ( !pin.isValidated() )
        ISOException.throwIt(ISO.SW_PIN_RIQUIRED);
    byte octetsLus = apdu.setIncomingAndReceive();
    if ( octetsLus != 1 )
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    balance = (byte) (balance + buffer[ISO.OFFSET_CDATA]);
}
```

```
// Réception de données
private void debiter( APDU apdu ) {
    if ( !pin.isValidated() )
        ISOException.throwIt(ISO.SW_PIN_RIQUIRED);
    byte octetsLus = apdu.setIncomingAndReceive();
    if ( octetsLus != 1 )
        ISOException.throwIt(ISO.SW_WRONG_LENGTH);
    if ( (balance - buffer[ISO.OFFSET_CDATA]) < 0 )
        ISOException.throwIt(BalanceNegative_SW);
    balance = (byte) (balance - buffer[ISO.OFFSET_CDATA]);
}
```


Programming....

```
// Émission de données
private void lire( APDU apdu ) {
    if ( !pin.isValidated() )
        ISOException.throwIt(ISO.SW_PIN_REQUIRED);
    apdu.setOutgoing();
    apdu.setOutgoingLength((byte)1);
    buffer[0] = balance;
    apdu.sendBytes((short)0, (short)1) ;
}

// Manipulation du code secret
private void valider( APDU apdu ) {
    byte octetsLus = apdu.setIncomingAndReceive();
    pin.check(buffer, ISO.OFFSET_CDATA, octetsLus);
}

}
```

Programming....

```
private DESKey myDESKey;

public static void install(byte[] bArray,
                           short bOffset, byte bLength) {
    new Encryption ();
    pin.updateAndUnblock(bArray, bOffset, bLength);
}

public boolean select() { pin.reset(); return true; }

public void process( APDU apdu ) {
    buffer = apdu.getBuffer();
    if ( buffer[ISO.OFFSET_CLA] != 0x00 )
        ISOException.throwIt(ISO.SW_CLA_NOT_SUPPORTED);
    switch ( buffer[ISO.OFFSET_INS] ) {
        case ENCRYPT_INS : encrypt(apdu); return;
        case PINCHECK_INS : pinCheck(apdu); return;
        default:
            ISOException.throwIt(ISO.SW_INS_NOT_SUPPORTED);
    }
}
```

Programming....

- Classe dérivant de `javacard.framework.Applet`
- Une applet carte est un programme serveur de la Java Card
 - APDU de sélection depuis le terminal (select)
 - Sélection par AID (chaque applet doit avoir un AID unique)
 - AID
 - 5 octets identifiant le propriétaire
 - 0-11 octets dépendant du propriétaire

- Classe dérivant de `javacard.framework.Applet`
- Une applet carte est un programme serveur de la Java Card
 - APDU de sélection depuis le terminal (select)
 - Sélection par AID (chaque applet doit avoir un AID unique)
 - AID
 - 5 octets identifiant le propriétaire
 - 0-11 octets dépendant du propriétaire

■ Cycle de vie : Méthodes appelées par JCRE

■ `static void install(bArray, bOffset, bLength)`

- Crée une instance de la classe avec les paramètres passés dans bArray
- Puis l'enregistre (register()) auprès du JCRE

■ `boolean select()`

- Appelé à la sélection
- peut retourner false si l'initialisation est incomplète (liaison impossible vers des objets partagés, ...)

■ `void deselect()`

- Appelé à la désélection

■ `void process(APDU apdu)`

- Méthodes appelées par JCRE

Version 2.2 de l'API de Java Card

| Package Summary | |
|---|---|
| Packages | |
| <code>java.io</code> | A subset of the <code>java.io</code> package in the standard Java programming language. |
| <code>java.lang</code> | Provides classes that are fundamental to the design of the Java Card technology subset of the Java programming language. |
| <code>java.rmi</code> | The <code>java.rmi</code> package defines the <code>Remote</code> interface which identifies interfaces whose methods can be invoked from card acceptance device (CAD) client applications. |
| <code>javacard.framework</code> | Provides a framework of classes and interfaces for building, communicating with and working with Java Card applets. |
| <code>javacard.framework.service</code> | Provides a service framework of classes and interfaces that allow a Java Card applet to be designed as an aggregation of service components. |
| <code>javacard.security</code> | Provides classes and interfaces that contain publicly-available functionality for implementing a security and cryptography framework on Java Card. |
| <code>javacardx.crypto</code> | Extension package that contains functionality, which may be subject to export controls, for implementing a security and cryptography framework on Java Card. |

EclipseJCDE – la vie devient facile

Rendez-vous au plus à cet url pour les windowsiens : (– <http://eclipse-jcde.sourceforge.net/>)

