Sécurité des cartes à puces et Java

IFT-21940

Cryptographie et sécurité informatique

19 mars 2002, Université Laval

Marco Savard, développeur Java

Travail pratique numéro 3

- Java et la sécurité
- Les smart cards
- Projet du TP
- Partie cardlet
- Partie client

Java et la sécurité

- La technologie
- Le langage
- Les bibliothèques de classes
- Les outils
- Les failles

La technologie

- C et C++: compilation en un .exe
- Java : compilation en un .class, interprété par un machine virtuelle Java (MVJ)
- La MVJ peut refuser d'exécuter un commande qu'elle juge non-sécuritaire

Le langage

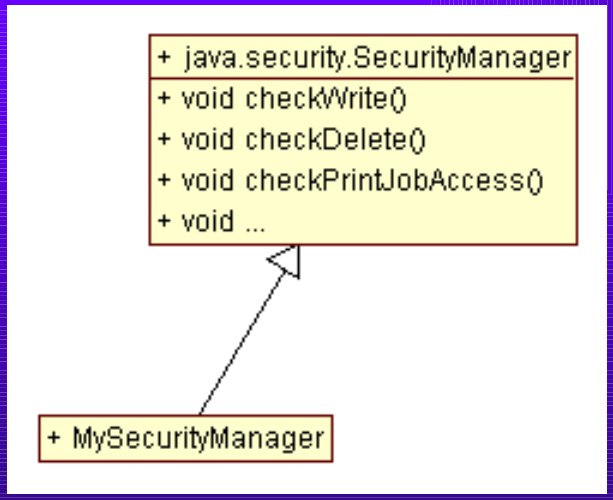
- Pas d'arithmétique de pointeurs
- Débordement de tableaux interdit
- Les chaînes sont immutables
  - String ip = '199.199.199.199';
  - ip[3] = '1' interdit
- Visibilité (private, protected) pour restreindre l'accès à des membres

Les bibliothèques

- Paquetage java.security:
  - SecurityManager
    - Implante un gestionnaire de sécurité sur mesure
  - Permission et ses sous-classes
    - Accès à une ressource
  - KeyPairGenerator
    - Génère clés privé et public (algo DSA ou autre)
  - SecureRandom
  - Signature
  - MessageDigest
    - Méthode de hachage

SecurityManager

- Implanter un gestionnaire de sécurité sur mesure
- System.setSecurityManager(mySM);



Les outils du JDK

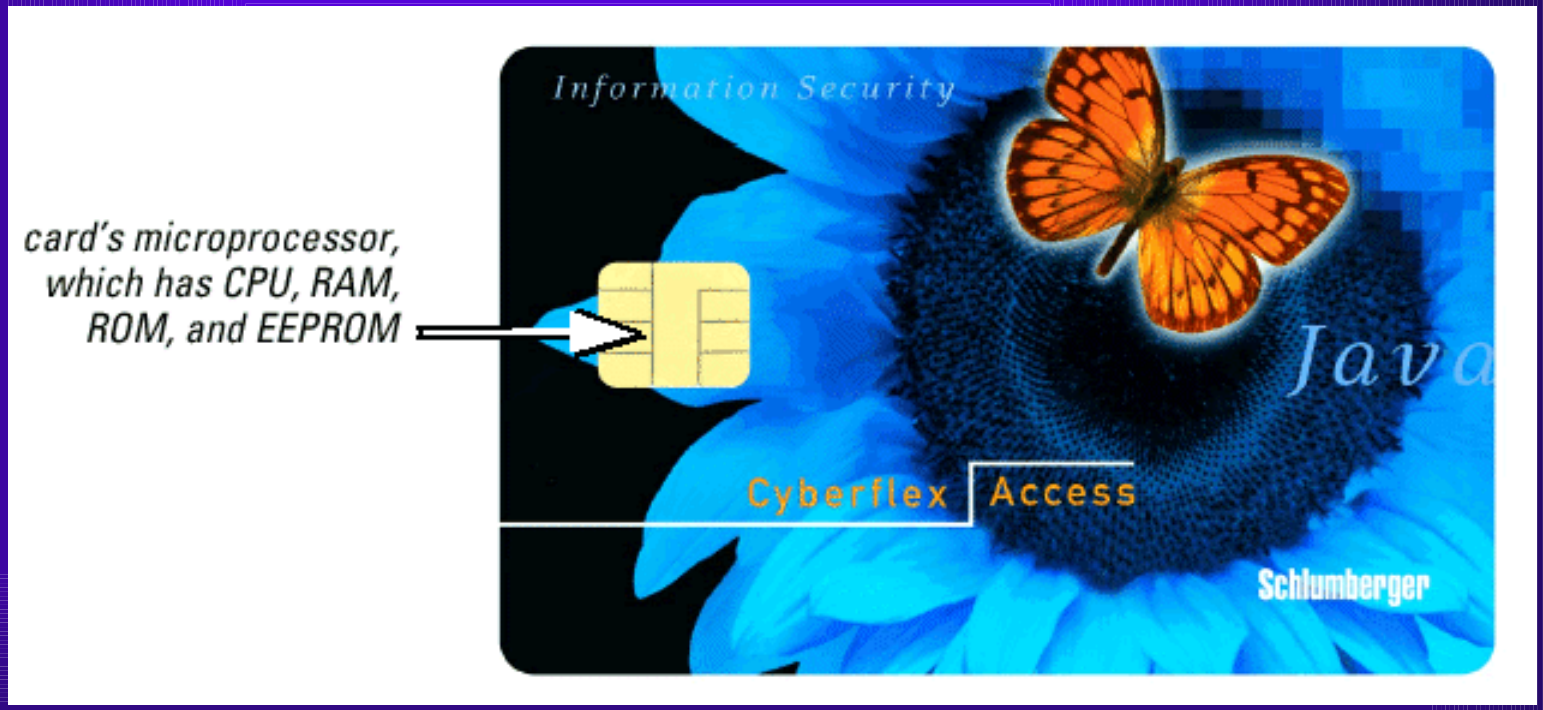
- Keytool (javakey avant 1.2)
  - Gestion des clés privés et publiques, export, génération, accès, certificats X.509
- Jarsigner
  - Jarsigner –keystore ‘userstore’ ‘dest’ ‘src’
- Policytool
  - Démarre un application gui pour définir des politiques (droit d’un usager à une ressource)
  - Java –Djava.security.manager ‘policyfile’ Main.class

Failles de Java

- Quelques failles connues;
- Par exemple, un classe interne accède à tous les membres de la classe externe.
- Le compilateur crée des fichiers séparés:
  - Outer.class, Outer$1.class Outer$2.class
- Possibilité d'ajouter un Outer$3 traffiqué

Smart Cards

- La smart card de Schlumberger

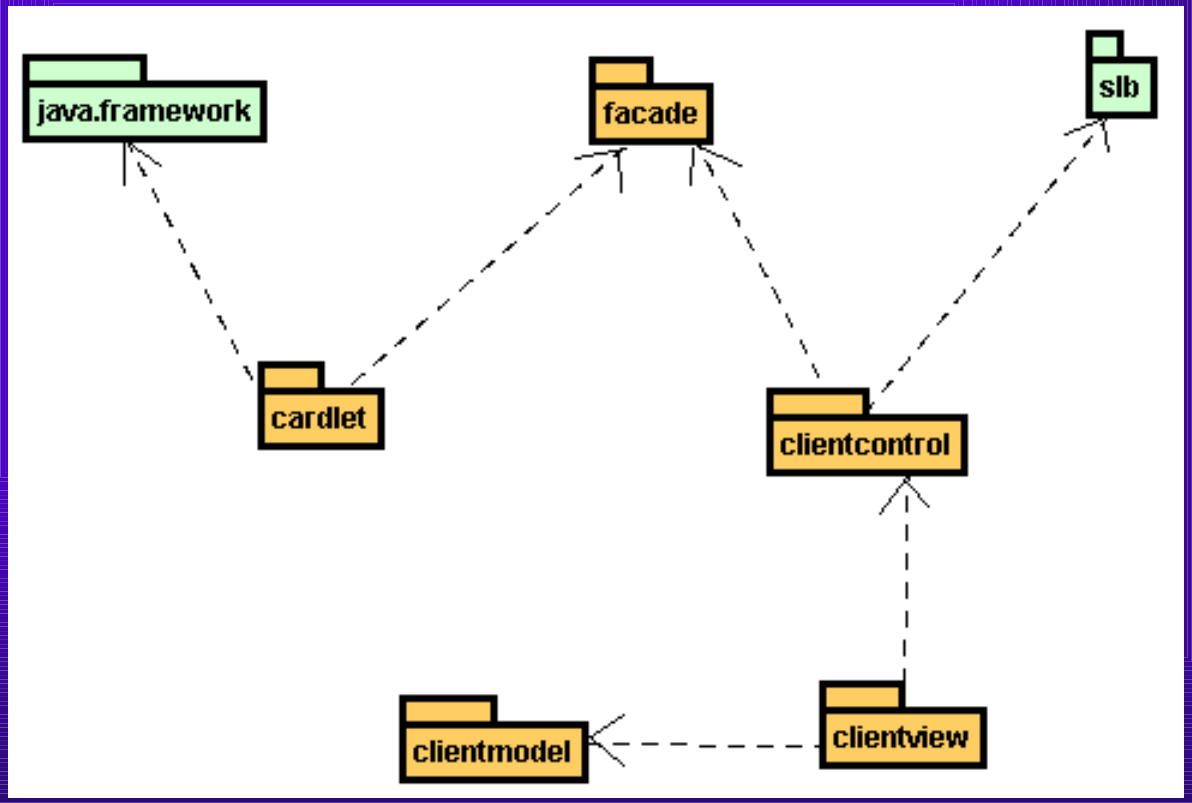


Projet

- Implanter un cardlet
- Compléter un client
- Sécuriser le client

Architecture

- Diagramme des paquetages:



Livrables

- Analyse: 10%
  - Détailler le use case, diagramme des classes UML
- Cardlet: total 30%
  - Implantation du code 25%; plan de test 5%
- Client: total 60%
  - Implanter les commandes 20%
  - Proposer façons de corriger les failles 10%
  - Corriger les failles 20%
  - Analyse d'autres failles 5%
  - Plan de test 5%

Partie Cardlet

- Liste des opérations de base du cardlet
- Détailler les ‘use-case’
- Implanter cinq opérations de base
- D’un source java à un cardlet
- Tester le cardlet avec APDU Manager

Liste des opérations de base

- Cinq opérations à implanter:
  - Validate() (INS 0x10)
  - LogAdmin (INS 0xF0)
  - SetCurrentAccount (INS 0x40)
  - GetCurrentAccount (INS 0x50)
  - PinChange (INS 0x30)
- Déclarer les autres opérations de façon à couvrir tous les ‘use cases’ (pas à implanter). S’inspirer de SignOnConstants.java

Détailler les ‘use-case’

- OuvrirSessAdmin
  - LogAsAdmin (0xF0)
  - Validate (0x10)
- AjouterCompte
  - Trouvé = faux;
  - Pour i de 1 à MAX_COMPTE
    - SetAccount i (0x40)
    - GetAccountID
    - Si accountID = 0
      - Trouvé = vrai;
      - Sortir
  - Si Trouvé = vrai
    - setAccountID matricule

Analyse

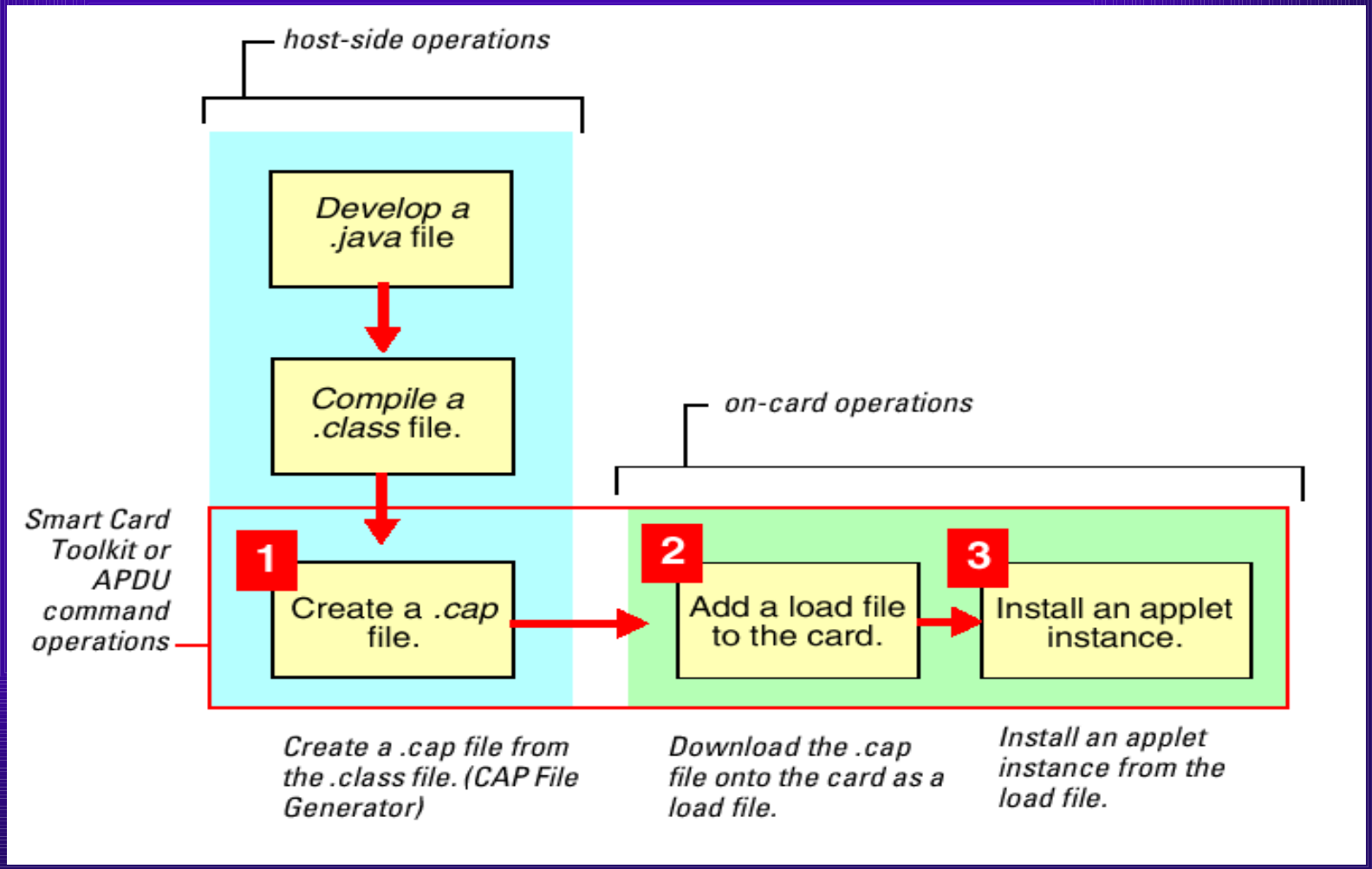
- Document ‘détail des use-cases’.
- Produire un diagramme de classes pour représenter les données décrites à la section ‘description des données’ du document de spécification (2 ou 3 classes).

Implanter les cinq opérations de base

- Compléter le corps des cinq méthodes dans le fichier SignOnApplet.java
- Compiler pour produire le fichier .class

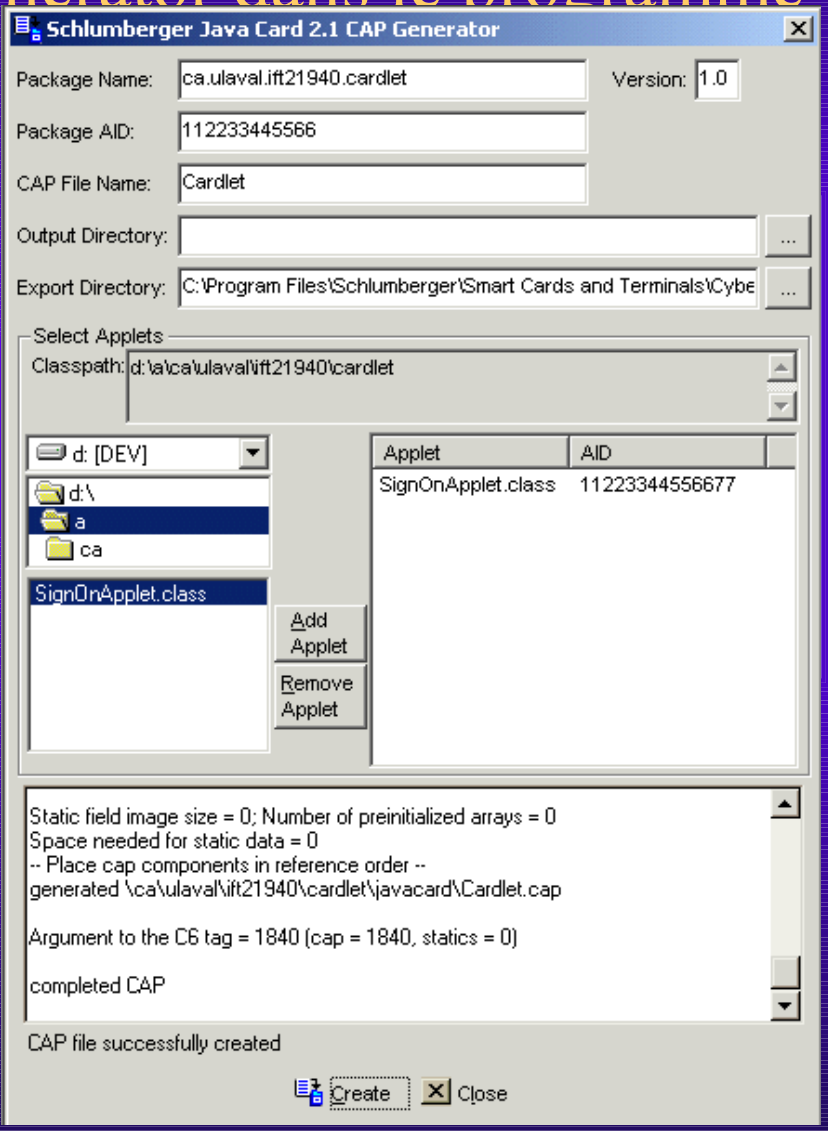
D'un .java à un .cap

(Cyberflex Programmer's Guide, page 56)



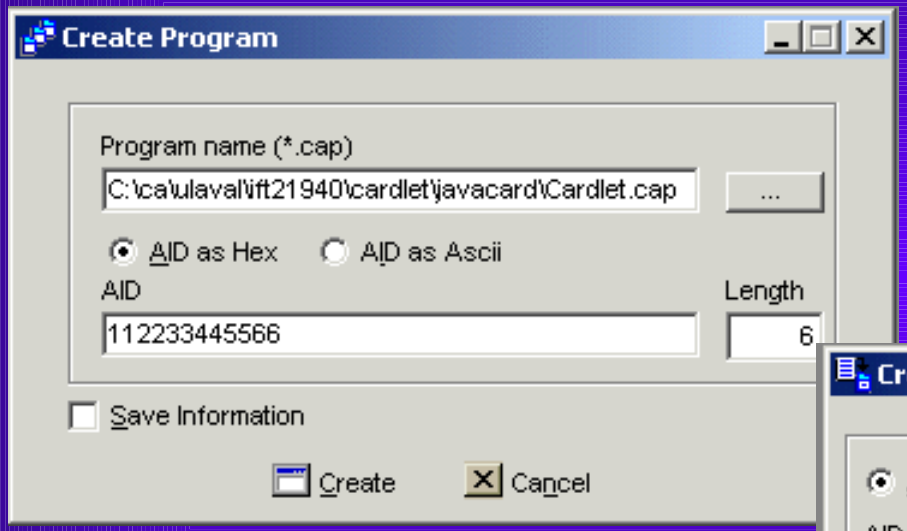
Générer le fichier .cap

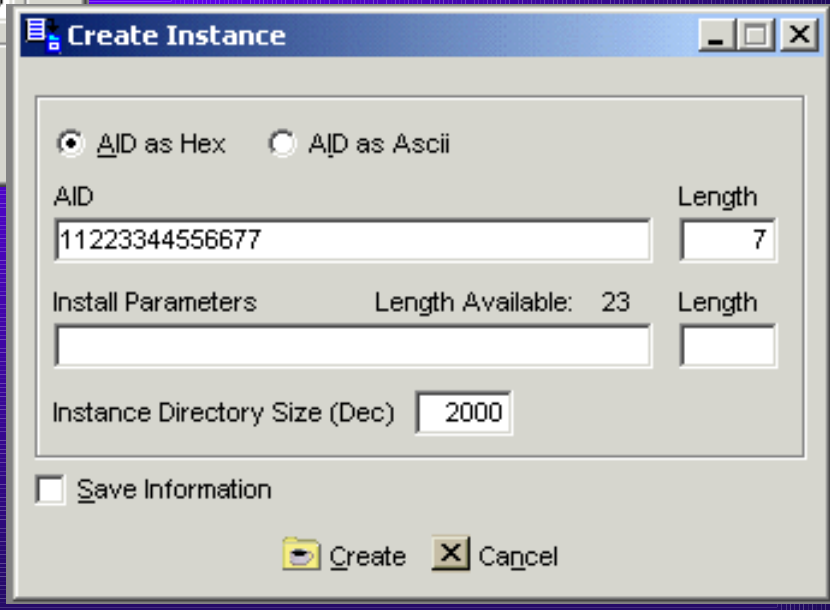
(CapFileGenerator dans le programme Toolkit)



Charger le cardlet dans la carte

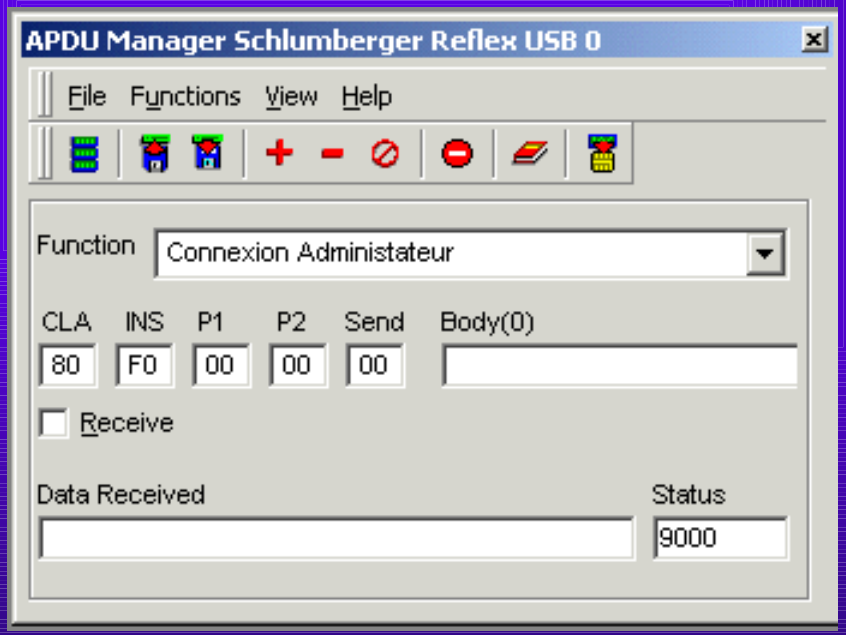
- Créer le programme
- Créer l'instance





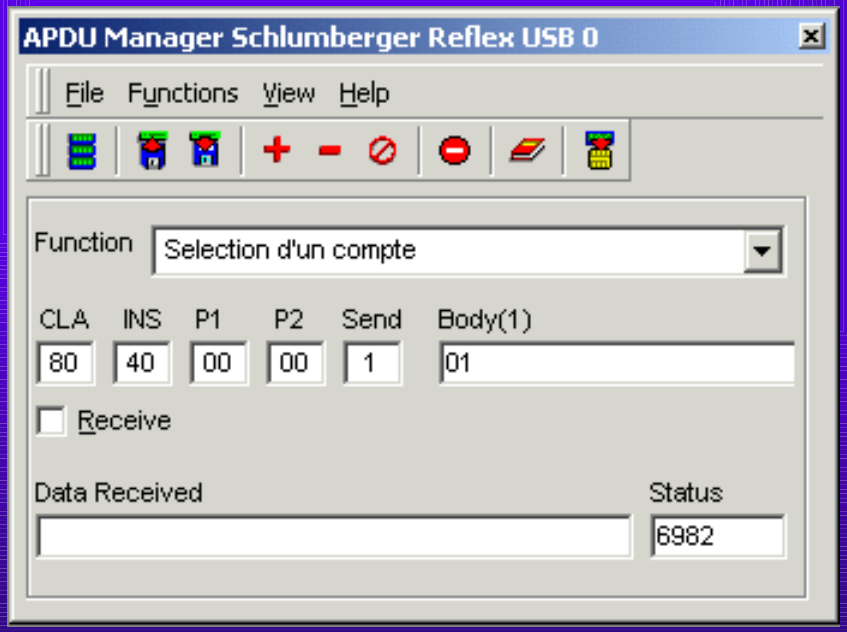
Tester le cardlet
(Utiliser le APDU Manager du toolkit)

- Connexion administrateur
  - Code de retour 9000 = succès



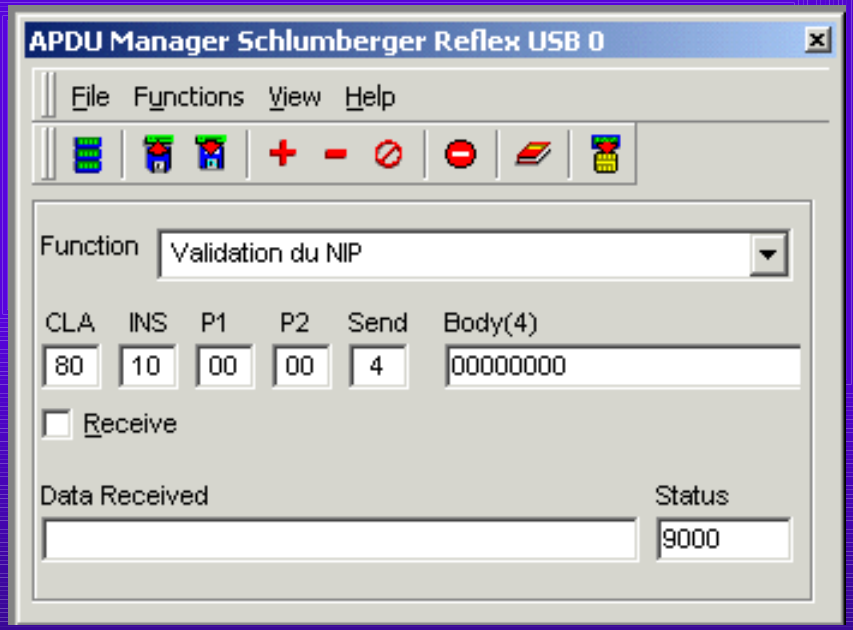
Tester le cardlet

- Sélection du compte numéro 1
  - Retourne code 6982 (voir CyberflexPG)



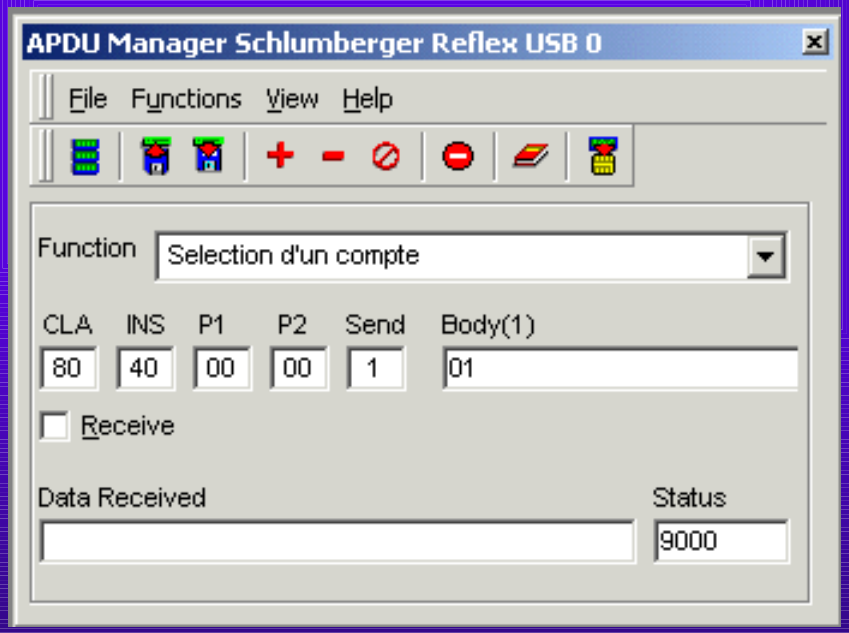
Tester le cardlet

- Entrer le NIP de l'administrateur



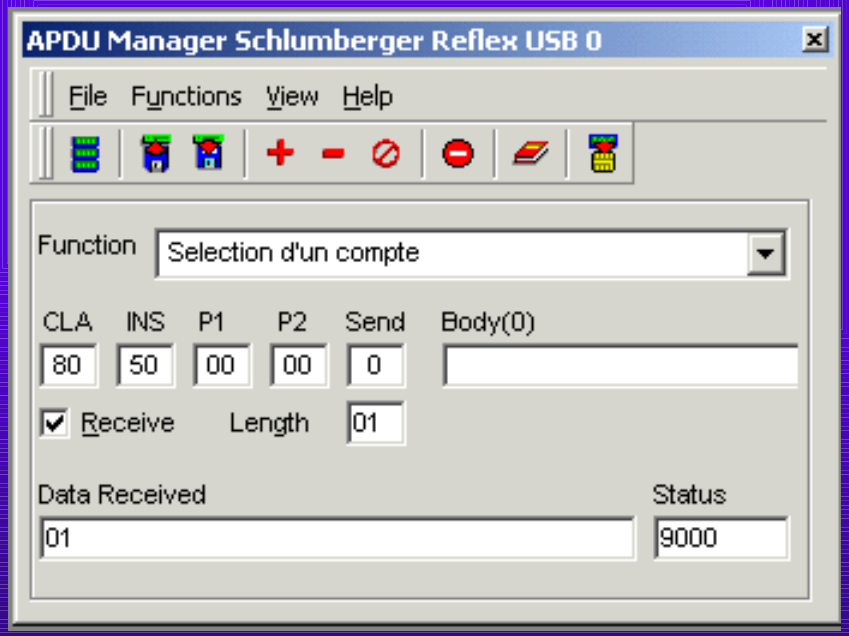
Tester le cardlet

- Sélection du compte numéro 1



Tester le cardlet

- Obtenir compte courant
  - Reçoit 01 (compte numéro 1)



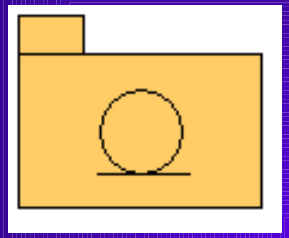
Normes de programmation des cardlets

- Classe héritant de Applet (celui de javacard.framework)
- Type primitifs: boolean, byte et short
- Pas de classes du langage (ex: String)
- Aucun fil d'exécution, de ramasse-miettes
  - Consulter CyberflexPG pour plus de précisions

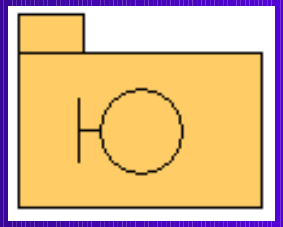
Partie client

- Patron de conception MVC
- Actions et commandes
- Implantation des commandes
- Tester les commandes (tests unitaires)
- Tester le client (test d'intégration)
- Sécuriser le client

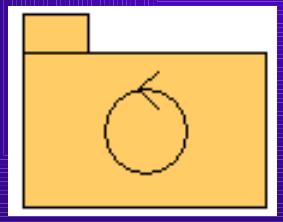
MVC



- Modèle : données persistantes
  - Package clientmodel (table des appl.)
  - Le cardlet lui-même!



- Vue : Interface avec l'usager (gui ou autre). Packages clientview.



- Contrôle : Traitement (commandes)

Actions et commandes

- Actions (partie vue):
  - Héritant de swing.AbstractAction
  - Rattachée à un élément graphique (bouton,ect)
  - Une seule occurrence par session
- Commandes (partie contrôle):
  - Patron de conception GoF95
  - Implémentent toutes une méthode execute()
  - Plusieurs occurrences par session, créées par l'action correspondante
  - Peuvent être conservées dans un CommandHistory

Tester les commandes

- Trois commandes à implanter :
  - Login, ChangePIN et AddAccount
- Pour ces commandes, faire un main()
  - Qui crée une occurrence de commande
  - Qui appelle la méthode execute()
  - Qui valide le résultat

Tester le client

- Invoquer le main() de MainFrame dans clientview.main
- Vérifier si les commandes sont appellés

Sécuriser le client :

4 failles à corriger

- Cacher graphiquement le NIP
- Désactiver l'option copier-coller
- Signer la table des applications (fichier .ser)
- Signer le client (.jar)

Sécuriser le client

- Pour chaque faille:
  - Documenter comment corriger la faille
  - Apporter les corrections dans le code source, s'il y a lieu.

Analyse d'autres risques

- Documenter les autres failles potentiels, en plus des quatre déjà indentifiées.
- Proposer une façon de corriger ces failles

Références

- Programmer's Guide de Schlumberger (documents PDF)
- Security in Java 2
  - http://java.sun.com/docs/books/tutorial/security1.2/index.html
- Java Examples in a Nutshell
  - http://www.oreilly.com/catalog/jenut2/

- FIN -