

Java CardTM

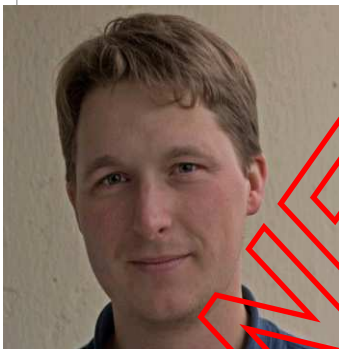
Damien Sauveron

damien.sauveron@unilim.fr

<http://damien.sauveron.fr/>

Intervenants dans le module

- Damien Sauveron (Cours, TD et TP) :
 - Membre de l'équipe « Cryptis » en charge du projet « Sécurité des Systèmes et Réseaux » (P.F. Bonnefoi, E. Conchon, K. Tamine)
 - Thème de recherche : sécurité des systèmes embarqués et des réseaux mobiles



- **Pour me contacter :**

Damien Sauveron
XLM UMR 7252 CNRS -- Université de Limoges
123 avenue Albert Thomas
87060 Limoges Cedex, FRANCE

Email: Damien.Sauveron@unilim.fr
Web: <http://damien.sauveron.fr/>
Phone: +33 (0) 5 87 50 67 93

- **MCF à l'Université de Limoges depuis septembre 2006 – HDR (2014)**
- **Visite postdoctorale au Smart Card Centre de l'Information Security Group du Royal Holloway University of London**
- **Thèse au LaBRI – Université Bordeaux 1 sur la Sécurité de la Technologie Java Card**
- **Ingénieur R&D dans le CESTI de SERMA Technologies (Pessac)**
- **Organisateur de WISTP2007 (Héraklion) et WISTP2008 (Séville)**
- **Président du groupe de travail de l'IFIP : Pervasive Systems Security**

Agenda

- La machine virtuelle Java
- La technologie Java Card
 - Introduction
 - Architecture
 - Mémoire
 - Les différentes versions
 - JCRE
 - JCVM
 - APIs

État de l'art des cartes avant Java Card

- La carte à puce est un véritable ordinateur utilisé comme serveur portable et sécurisé de données personnelles
- Elle est programmée comme un composant embarqué avec un code applicatif figé
- Elle est difficile à intégrer dans les systèmes d'informations
 - Car si l'application requiert de nouvelles fonctions carte il faut développer un nouveau masque (=> coûteux)
 - => Sinon il faut prévoir un masque qui accepte d'exécuter des programmes en EEPROM (rare et pas standardisé)
 - Car pas d'interface standard de communication avec les différents lecteurs (travaux en cours – problème complexe)
 - Communication via APDUs (très bas niveau)
- 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 et accélérer les développements de codes dans la carte
 - Faire de la carte un environnement d'exécution de programmes ouvert (chargement dynamique de code)
 - Rendre plus souples et plus évolutives les applications carte
 - Faciliter l'intégration des cartes dans les applications
 - Faciliter et accélérer les développements d'applications clientes des cartes
- Besoins :
 - Diversification des sources cartes et des terminaux
 - Pérennité des développements
- Procédé de construction des applications
 - Permettre le développement «rapide» d'applications carte «évoluées» par des «non spécialistes»
 - But : de nouvelles applications pour de nouveaux marchés !

Qu'est ce que Java Card ?

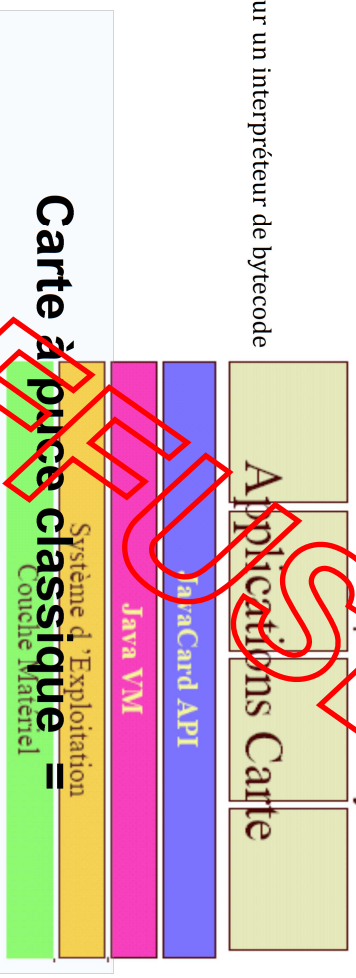
- Technologie permettant de faire fonctionner des applications écrites en langage Java pour :
 - carte à puce



- autres périphériques à mémoire limitée (par exemple JButton)



- De manière intuitive :
 - Il s'agit d'une carte basée sur un interpréteur de bytecode



- La technologie Java Card définit une plate-forme pour cartes à puce sécurisée, portable et multi-applicative qui incorpore beaucoup des avantages du langage Java.

Historique 1/2

- En Novembre 1996, un groupe d'ingénieurs de Schlumberger cherche à simplifier la programmation des cartes à puces tout en préservant la sécurité (Cyberflex 1.0 au CNIT - Cartes'96).
==> la spécification Java Card 1.0
- En Février 1997, Bull et Gemplus se joignent à Schlumberger pour cofonder le « Java Card Forum ».
 - Consortium de fabricants :
 - Cartes (Gemplus, Schlumberger, etc)
 - Informatique : IBM, Sun
 - Matériel : DEC, Motorola
 - Utilisateurs : Banques (Citibank, etc.)
 - Buts :
 - promouvoir Java Card
 - Faire des choix technologiques communs
 - Contre pouvoir à Sun (officieusement)
 - Solutions :
 - Un comité technique
 - Un comité business
 - <http://www.javacardforum.org/>

- En Novembre 1997, Sun présente les spécifications Java Card 2.0.
- 1998 : Cyberflex 2.0, GemXpresso, ...

Historique 2/2

- En Mars 1999 sort la version 2.1 des spécifications Java Card. Elles consistent en trois spécifications :
 - The Java Card 2.1 API Specification.
 - The Java Card 2.1 Runtime Environment Specification.
 - The Java Card 2.1 Virtual Machine Specification.
 - Contribution la plus significative :
 - Définition explicite de la machine virtuelle de la Java Card.
 - Le format de chargement des applets.
- En Mai 2000, sort une petite correction ==> version 2.1.1
- En Octobre 2000, plus de 40 entreprises ont acquis la licence d'exploitation de la technologie Java Card.
- En juin 2002, spécifications Java Card 2.2.
 - Introduction de RMI, des canaux logiques, de nouveaux algorithmes cryptographiques, ...
- En octobre 2003, spécifications Java Card 2.2.1.
- En novembre 2005, la version 2.2.2 est Public Review Draft
- En décembre 2005, les licenciés sont :
 - ARM, Aspects, Axalto, CCL/ITRI, Fujitsu, Gemplus, Giesecke & Devrient GmbH, GoldKey Technology, HiSmarTech, I'M Technologies Ltd., IBM, incard, KEBTechnology, Logos Smart Card, Microelectrónica Española, Oberthur Card Systems, ORGA Kartensysteme, SAGEM, Sermepa, Setec, Sharp, Smart Card Laboratory Inc., SSP Solutions, Inc., STMicroelectronics, Toppan Printing, Trusted Logic. + d'autres qui préfèrent rester anonymes.

JavaCard Forum Members

(as of 10th Jan 2005)



Giesecke & Devrient



DatacardGroup



aspects



ARM



HITACHI



Les avantages de la technologie Java Card

- La facilité de développement des applications} grâce :
 - à la programmation orientée objet offerte par Java
 - à l'utilisation des environnements de développement existants pour Java.
 - à une plate-forme ouverte qui définit des APIs et un environnement d'exécution standard.
 - à l'encapsulation de la complexité sous-jacente du système des cartes à puce.
- L'indépendance au hardware réalisée grâce au langage Java
 - ==> « *Write Once, Run Anywhere* »
- La gestion dynamique de multiples applications
 - ==> possibilité de mises à jour des applications de la Java Card sans avoir besoin de changer de cartes
 - ==> possibilité d'offrir des nouveaux services
- La compatibilité avec les standards existants pour les cartes à puce
 - ISO7816 :
 - Protocole de communication
 - Protocole de haut niveau pour la gestion de fichiers, etc



- On verra les inconvénients plus tard.

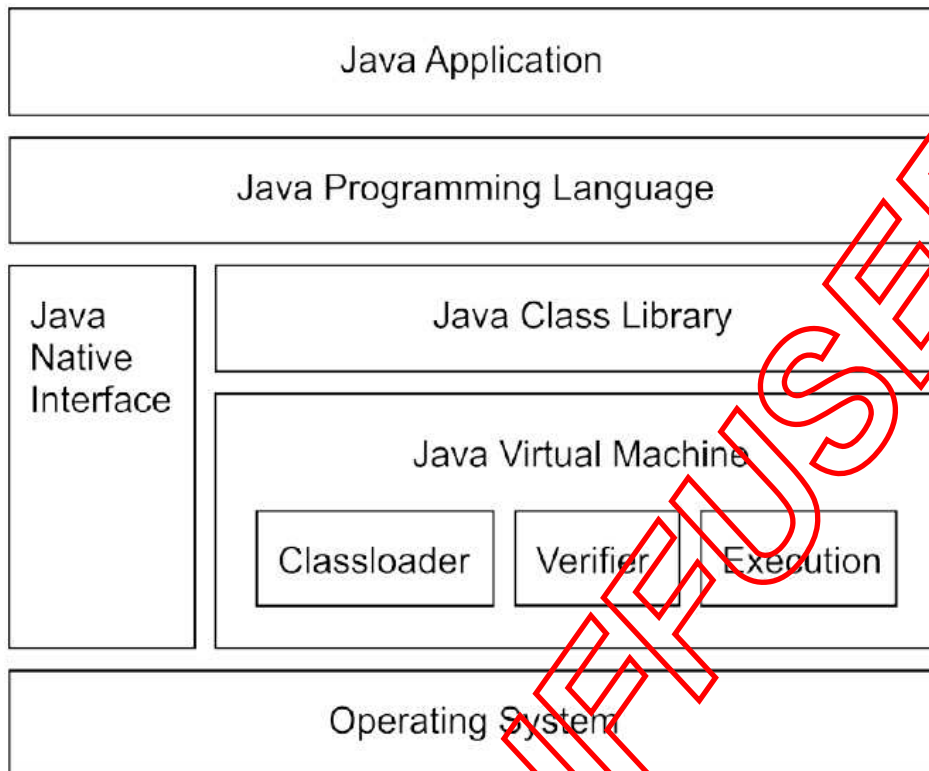
Java Card = Java + smart Card



La plate forme Java

- Java est un langage de programmation
 - Voir le « white paper » de J. Gosling
 - Un programme Java est compilé et interprété
- Java est une plate-forme
 - La plate-forme Java, uniquement software, est exécutée sur la plate-forme du système d'exploitation,
 - La « Java Platform » est constituée de:
 - La « Java Virtual Machine » (JVM)
 - Des interfaces de programmation d'application (Java API),
 - Une interface pour les méthodes natives (JNI)

Architecture de la plate-forme



Les différentes plates-formes



Les composants

- Le langage de programmation,
- La librairie,
- La machine virtuelle
 - Un jeu d'instruction interprétable (*byte codes*)
 - Un format de chargement, pas forcément équivalent au format d'exécution le *class file*
 - Un algorithme de vérification des règles de typage Java ramenées au byte code, obligatoire pour les applets,
 - Un compilateur à la volée.

Java langage de programmation

- Java est un langage de programmation particulier qui possède des caractéristiques avantageuses:
 - Simplicité et productivité:
 - Intégration complète de l'OO
 - Gestion mémoire (« Garbage collector »)
 - Robustesse, fiabilité et sécurité (langage fortement typé)
 - Indépendance par rapport aux plates-formes
 - Distribution et aspects dynamiques
 - Performance

Sécurité par le typage

- En Java les pointeurs existent mais:
 - la manipulation explicite d'un pointeur en tant que tel est impossible en Java **le compilateur vérifie** cette règle ainsi que la VM. Ainsi le code suivant n'est pas compilable:

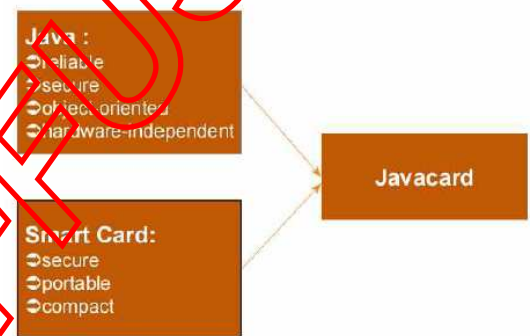
```
...  
p=new Object();  
p++;
```
 - tous les objets Java sont en fait des pointeurs (que l'on nomme références).
- Les variables locales (variables de types primitifs et références) sont allouées sur la pile (ou dans les registres) et les objets dans le tas.
- On ne contrôle pas la zone d'allocation ni quand sera effectuée la désallocation.

Sécurité par le typage

- Lors du chargement **la VM vérifie** si, au niveau byte code, il n'y a pas de transtypage ne correspondant pas aux règles de Java : **c'est le rôle du vérifieur de type.**
- Après, une référence n'a PLUS besoin de vérifier si le typage est correct.
- Donc la sécurité par le typage repose sur:
 - Un algorithme dans le compilateur,
 - Un autre dans la VM.

Java Card

Java Card = Java + smart Card

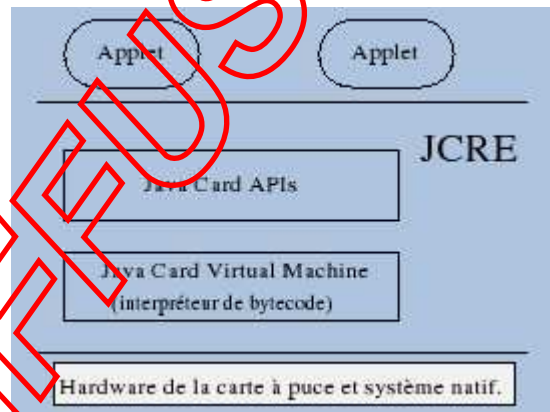


Attention

- Les informations concernent Java Card 3.0 Classic Edition (pas Connected Edition) et Java Card 2.X

Présentation générale

- Un ensemble de spécifications
 - Publiées par Oracle
 - Basées sur la plateforme Java
 - <http://www.oracle.com/technetwork/java/embedded/javacard/index.html>
- Découpées en trois parties
 - Application programming interfaces (APIs)
 - Execution environment (JCRE)
 - Virtual machine (VM)



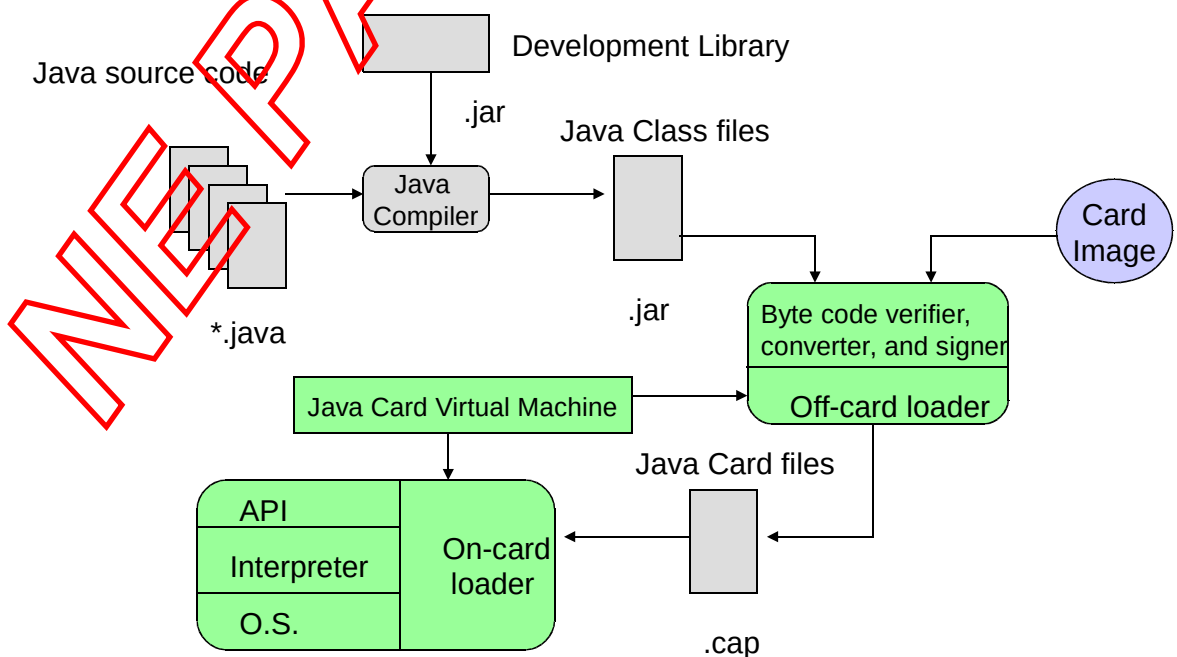
Agenda

- **Introduction**
- JCRE (Runtime Environment)
- JCVM (Virtual Machine)
- APIs

Un autre point de vue

- Architecture
- Mémoire
- Les différentes versions

L'architecture Java Card



Architecture

- Converter (part of the JCDK):
 - Class file format takes too much space on Java Card
 - Produces a format that fits the Smart Card constraints,
- Tokenization of the format
 - Need a representation of the content for pre-linking
 - The converter uses all class files of a package and all export files of ALL imported packages,
 - Output an Export file and a Cap file
- Conversion process
 - Verifies the Java Card language restriction
 - Optimize byte code
 - Invokes the off card verifier

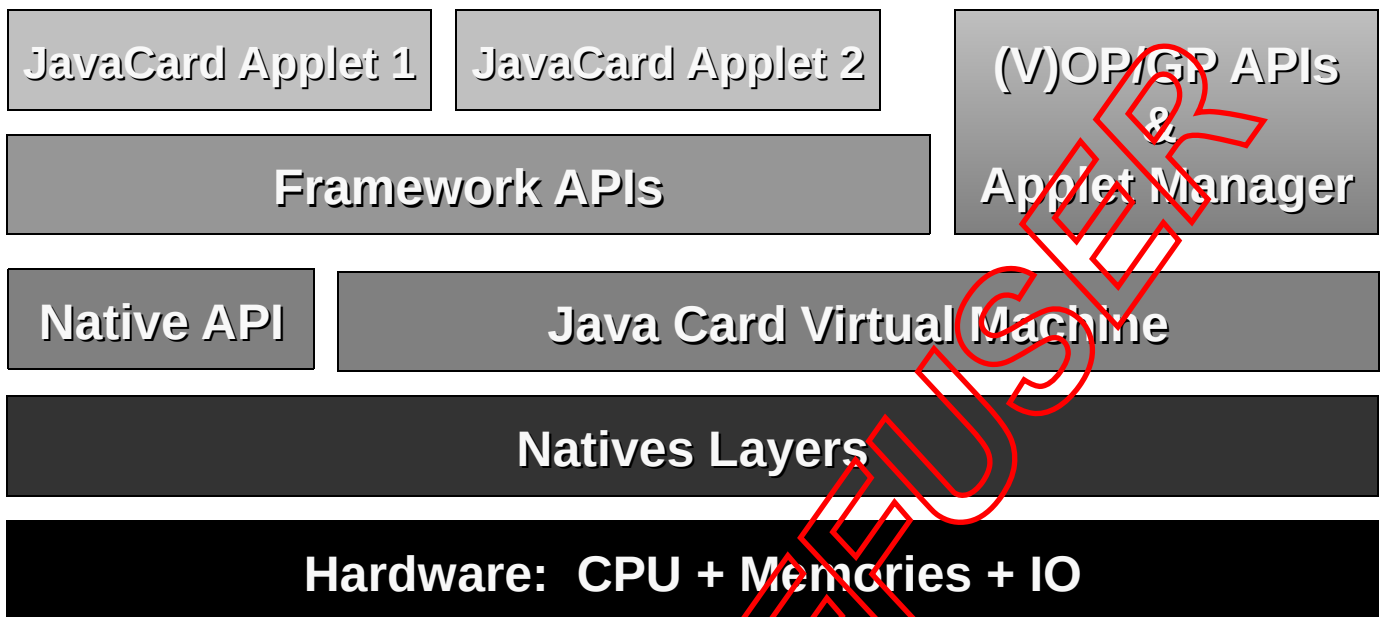
Two specific file formats

- The CAP (Converted Applet) file format
 - **Contains all the classes from one package**
 - Semantically, is equivalent to a set of class (.class) files
 - Syntactically, differs a lot from class (.class) files
 - All "string names" are replaced by "token identifiers"
- The EXP (Export) file format
 - Maintains the consistency between the originated class (.class) files and the resulting CAP file
 - Only for public (exported) data
 - Contains API information for a package of classes (access scope, class name, method signature, ...)
 - **Can be freely distributed**, used during pre-linking phase
 - Not loaded into the card

Interdependences

-
- The diagram illustrates the relationships between various JVM components. The central component is the **Method Component** (blue box). It is connected to several other components:
- Export Component** (orange box) and **Applet Component** (light blue box) are connected to the Method Component by solid lines.
 - Reference Location Component** (pink box) is connected to the Method Component by a solid line.
 - Static Field Component** (light blue box) is connected to the Method Component by a solid line.
 - Descriptor Component** (light blue box) is connected to the Method Component by a solid line.
 - Constant Pool Component** (green box) is connected to the Method Component by a solid line.
 - Import Component** (light blue box) is connected to the Constant Pool Component by a solid line.
 - Class Component** (yellow box) is connected to the Method Component by a solid line.
 - Directory Component** (white box) and **Header Component** (white box) are connected to the Class Component by solid lines.
- Dotted lines represent indirect or optional relationships:
- A dotted line connects the **Export Component** to the **Static Field Component**.
 - A dotted line connects the **Applet Component** to the **Static Field Component**.
 - A dotted line connects the **Reference Location Component** to the **Static Field Component**.
 - A dotted line connects the **Static Field Component** to the **Descriptor Component**.
 - A dotted line connects the **Descriptor Component** to the **Constant Pool Component**.
 - A dotted line connects the **Constant Pool Component** to the **Class Component**.
 - A dotted line connects the **Class Component** to the **Header Component**.

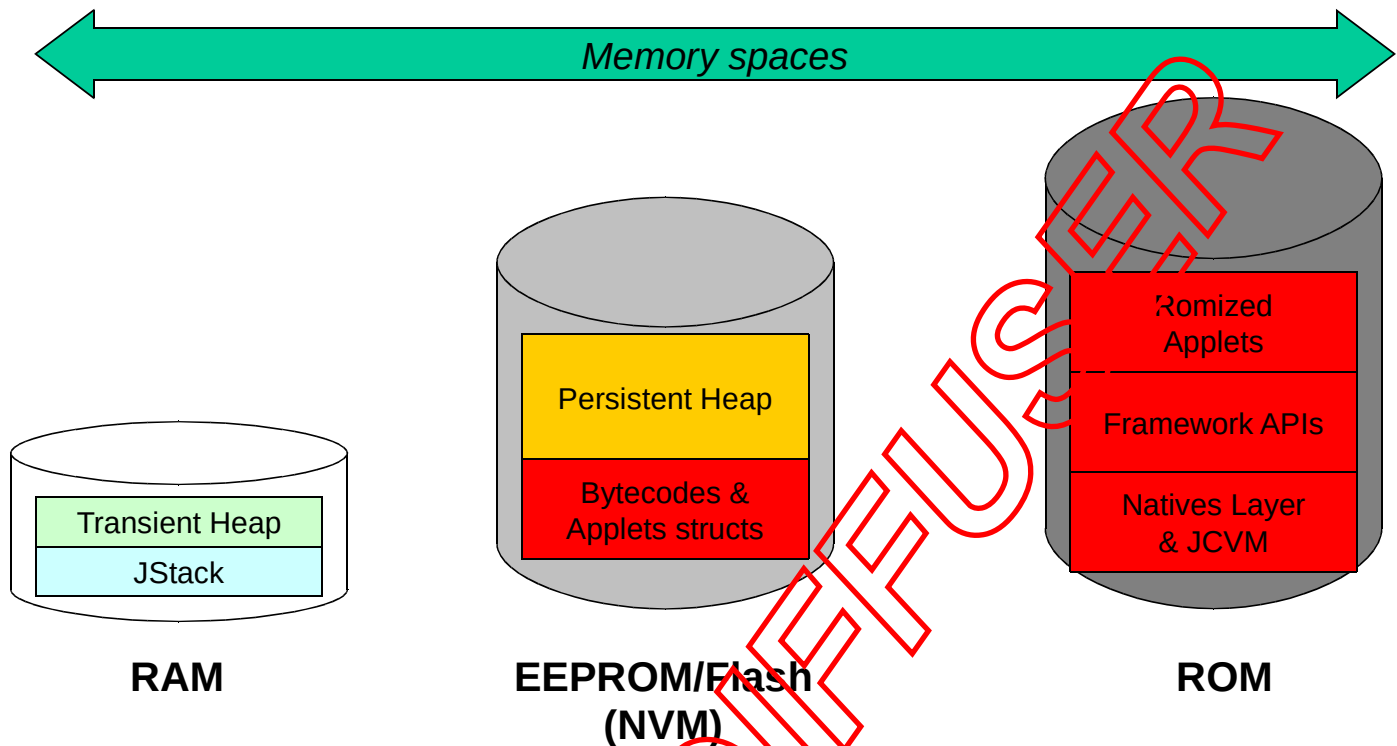
Java Card architecture



Java Card memory model

- By default, all objects are *implicitly* persistent
 - Because we have few RAM
 - Objects must survive between two sessions
- Some arrays can be *transient*
 - For efficiency and security reasons
- Transactional mechanisms are provided
 - All write operations on persistent memory are atomic
 - At the programming level a mechanism to handle transactions is also available

Java Card memory



Java Card 2.1

- File format of applet:
 - Standardized
 - file .cap identical to .class excepted:
 - One .cap file per java package
- « Firewall » between applets
 - The virtual machine must ensure that the code doesn't run out of its execution space (context)
 - Means to switch the execution context
 - Entry point object and global array can be accessed by applets (e.g., APDU)
 - JCRE can access each object
 - Interaction between applet through a shareable interface
 - System.share (Object ...) method suppressed

Java Card 2.2

- Load process of the .cap file standardized
 - Interoperability of JC 2.1 stops at the smart card loading,
- Object, applet and package deletion
- On card verifier (optional),
- Logical channels,
- Optional Garbage collector on demand,
- Support elliptic curves and AES algorithm,
- JC-RMI
 - Skeleton/stub generator 'a la RMI' hide the APDU encoding-decoding
 - Rapid development and integration of SC applets

Agenda

- Introduction
- JCRE (Runtime Environment)
- JCVM (Virtual Machine)
- APIs

Execution environment (JCRE)

- Define how a Java Card manages its resources
- Define constraints on the Java Card operating system
 - Applet lifetime (installation, register and deletion)
 - Logical channels and applet selection,
 - Transient objects,
 - Applet isolation (firewall) and sharing,
 - Transaction and atomicity,
- RMI
- The JCRE is at the heart of a Java Card

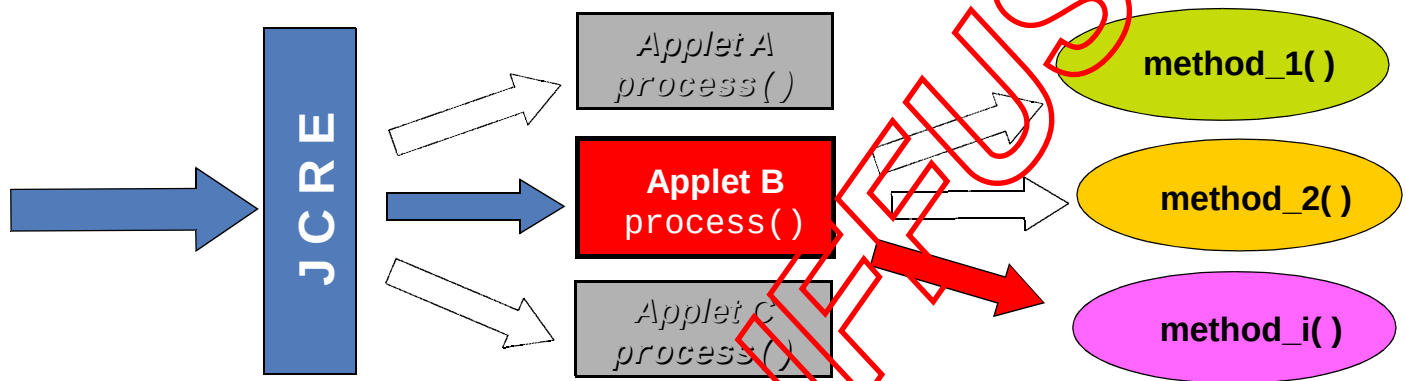
APDU commands

- 2 types of APDU can be sent to the card:
 - OS/Administrative commands
 - OS commands available in JCRE and CM
 - Select, Load, Install ...
 - Administrative commands specified by SC manufacturer
 - Get Info, ...
 - Applicative commands
 - specific to the JC applets loaded in the card
 - eg: debit, credit, getbalance for an e-purse applet
 - eg: create file, update file for the GSM applet



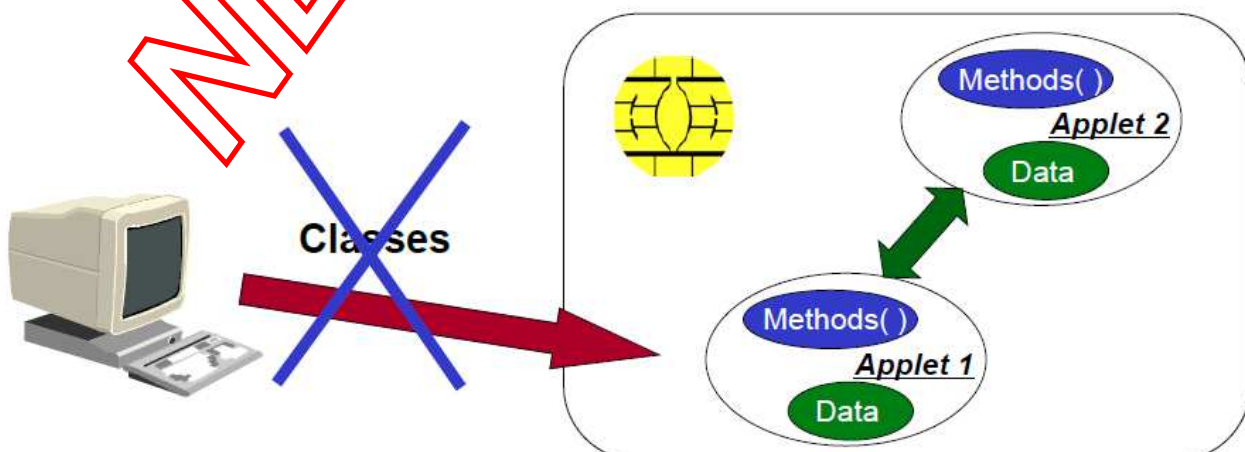
2 command dispatchers

- **JCRE's task:** main dispatcher
 - Route the incoming commands to the JCVI and the selected applet
- **Applet designer's task:** second dispatcher
 - Implement the applet's command dispatcher (extraction of the header information and call of the associated method)

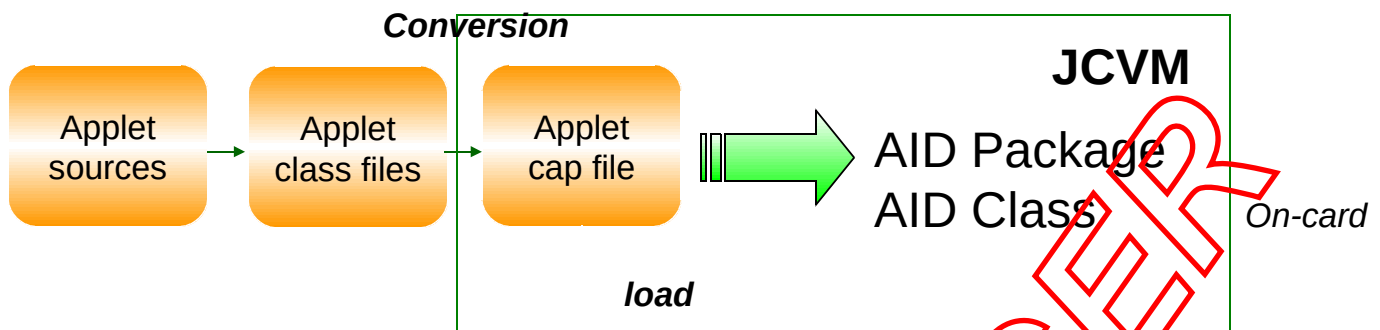


Unsupported features Dynamic class loading

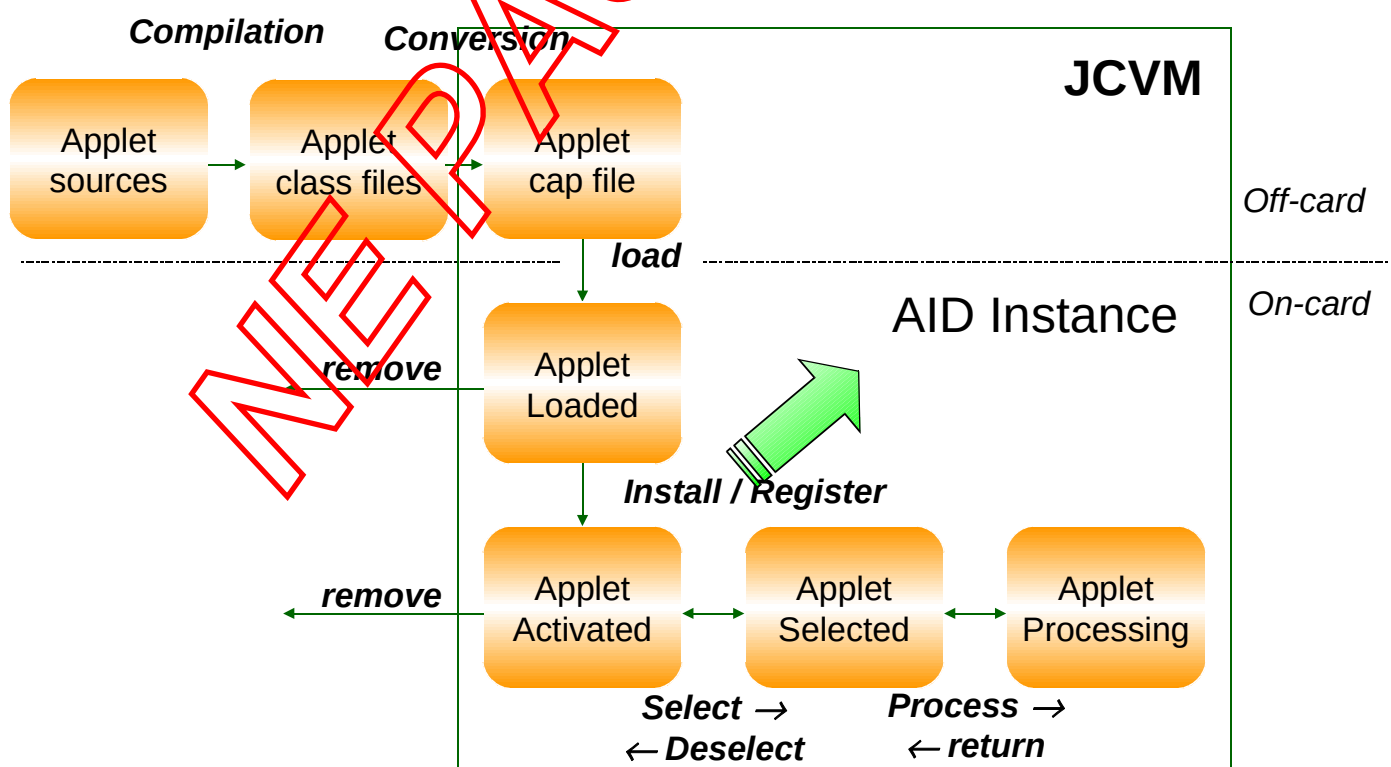
- Classes are **statically linked** before being downloaded
- No way to download classes on the fly as needed...
- Applets only refer to classes which already exist on the card



Java Card applet life cycle



Java Card applet life cycle

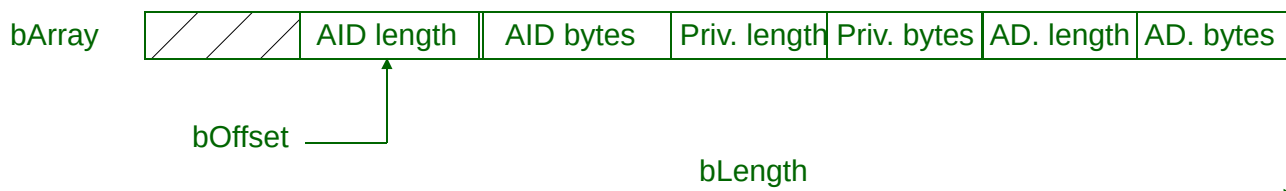


Methods of the applet

- An applet extends `javacard.framework.Applet`,
- It MUST implement the method `install`
- It MUST call the method `register`
- It MUST implement the method `process`
- It SHOULD implement `select`, `deselect`
- The `process` method can only be called if the applet has been selected

`install` method

- JCRE calls this static method first and gives
 - Applet instance AID
 - Applet privilege
 - Applet parameters
- `install` method creates an instance of an Applet subclass
 - Performs any necessary initializations,
 - Must call once one of the `register` methods
 - If no parameter is provided, only one installation



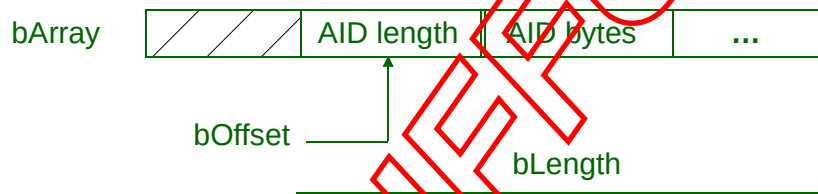
register method

- Used by the applet to register this applet instance
- Interacts with the Java Card Runtime Environment
 - `final void register()`
 - Assign the applet instance AID with default AID bytes defined in CAP file
 - `final void register(byte [] bArray, short bOffset, byte bLength)`
 - Assign applet instance AID with the specified AID bytes
- Warning: when receiving the byte array as parameters of the install command, the length is sent before the AID

```
public static void install(byte[] bArray, short bOffset, byte bLength) throws ISOException {
```

=>

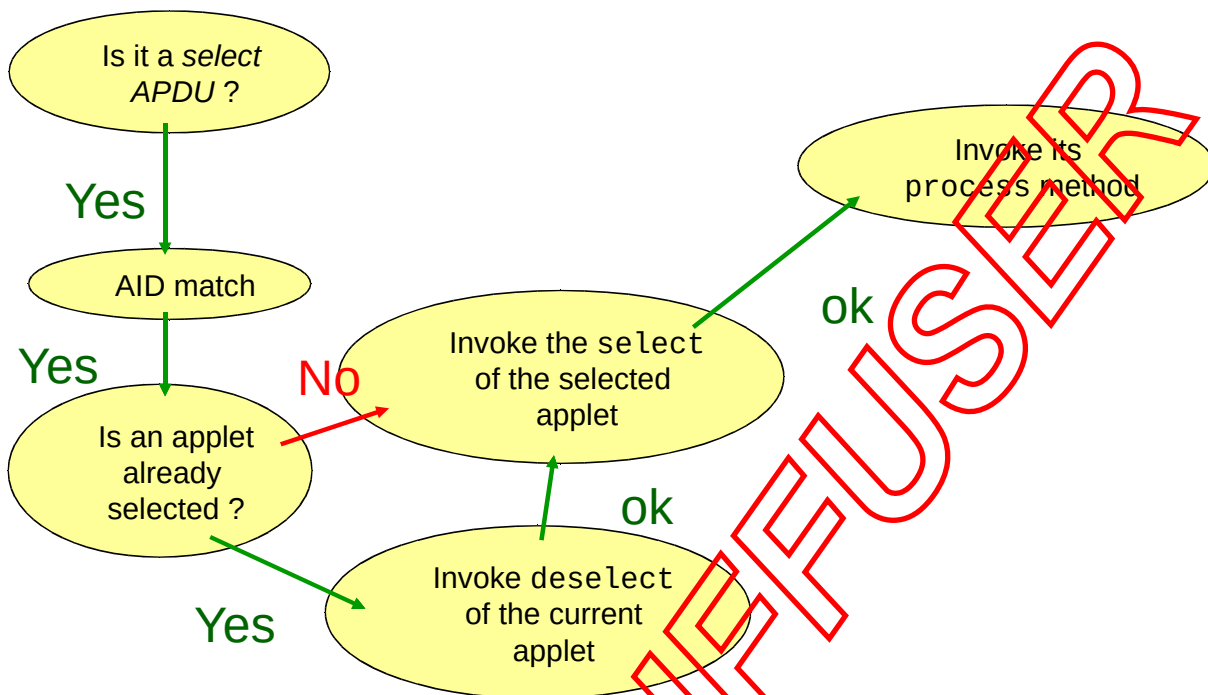
```
register(bArray, (short)(bOffset + 1), (byte)bArray[bOffset]);
```



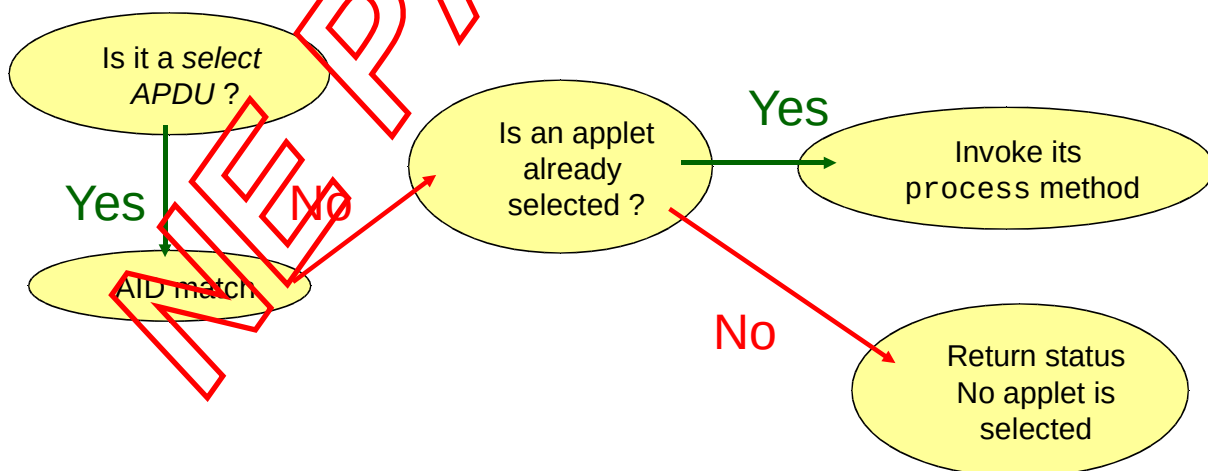
process method

- Contains the core application code of the applet
- Handles all the incoming APDU messages for the applet selected
- Called by the JCRE
- Upon normal return from this method the Java Card runtime environment sends the ISO 7816-4 success status word 90 00
- If it throws an exception the JCRE sends the associated reason code as the response status

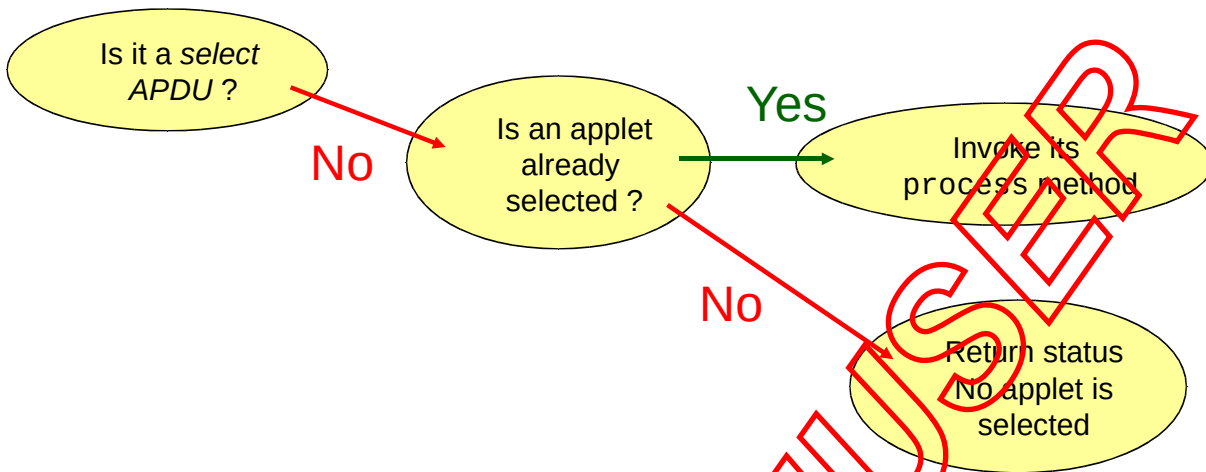
Selection protocol



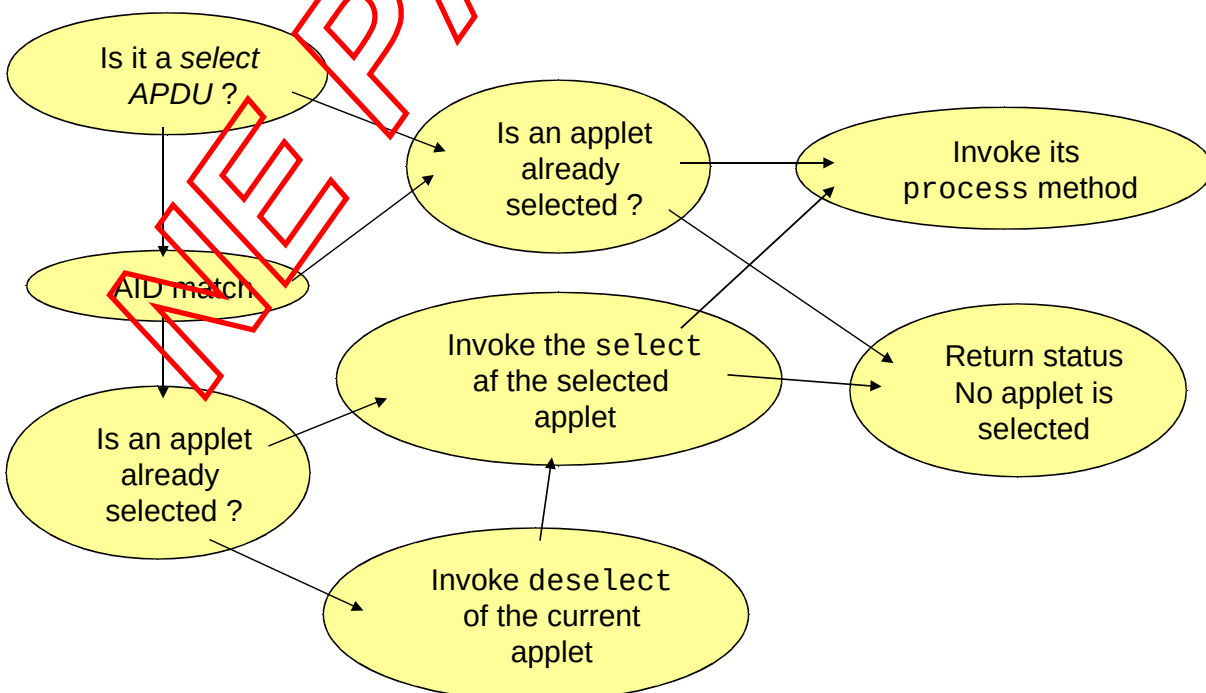
Selection protocol



Selection protocol



Selection protocol



Agenda

- Introduction
- JCRE (Runtime Environment)
- JCVM (Virtual Machine)
- APIs

Virtual machine (VM)

- This specification is made of several parts
 - The definition of the Java language subset that is supported
 - The definition of 2 specific file formats
 - The definition of a specific instructions set

Java language subset

- Supported
 - boolean, byte, short
 - int (optional)
 - Objects
 - Arrays
 - Virtual methods
 - Dynamic allocation
 - Packages
 - Exceptions
 - Interfaces
- Not supported
 - float, double, long
 - char, String
 - Multi-dimensional arrays
 - Garbage collector
 - Finalization
 - Threads
 - Dynamic loading of classes
 - Security manager

BC interpretation

- It is the execution engine for the byte code loaded into the card,
- It controls byte code execution, memory allocation and participate to the security through the firewall,
- Often it includes more tests than the firewall due to the absence of BC verifier...

Note that hardware processors executing Java Card bytecode exist (e.g. ST22).

Agenda

- Introduction
- JCRE (Runtime Environment)
- JCVM (Virtual Machine)
- **APIs**

APIs (1/2)

- Define a set of Java classes
 - Used to develop Java Card services (applets)
 - Dedicated for a smart card environment
- Those classes comply with current standard
 - Business Java (**java.lang.Object**)
 - ISO-7816 (*e.g.*, APDUs)
 - Cryptographic
- Those classes do not define by themselves services

APIs (2/2)

- Contains 3 mandatory packages
 - **java.lang**
 - Basic classes of the language
 - **javacard.framework**
 - Framework of classes and interfaces for the core functionality of an applet (AID, APDU, Applet, ISO7816, PIN, OwnerPIN, JCSystem, Util, and exceptions classes)
 - **javacard.security**
 - Core classes dedicated to cryptographic services (public/private key, random number generator,...)
- And one optional package
 - **javacardx.crypto**
 - Implementation classes for ciphering/deciphering (strong cipher)

java.lang

- **Object** { **public Object()**,
public boolean equals(Object obj); }
- **Throwable** { **public Throwable();** }
 - **Exception**
 - **RuntimeException**
 - **ArithmeticException**
 - **ClassCastException**
 - **NullPointerException**
 - **SecurityException**
 - **ArrayStoreException**
 - **NegativeArraySizeException**
 - **IndexOutOfBoundsException**
 - **ArrayIndexOutOfBoundsException**

javacard.framework

- class java.lang.Object
 - class javacard.framework.AID
 - class javacard.framework.APDU
 - class javacard.framework.Applet
 - class javacard.framework.JCSystem
 - class javacard.framework.OwnerPIN (implements javacard.framework.PIN)
 - class java.lang.Throwable
 - class java.lang.Exception
 - class javacard.framework.CardException
 - class javacard.framework.UserException
 - class java.lang.RuntimeException
 - class javacard.framework.CardRuntimeException
 - class javacard.framework.APDUException
 - class javacard.framework.ISOException
 - class javacard.framework.PINException
 - class javacard.framework.SystemException
 - class javacard.framework.TransactionException
 - class javacard.framework.Util
 - interface javacard.framework.ISO7816
 - interface javacard.framework.PIN
 - interface javacard.framework.Shareable
- } Optionally used by the developer
- class javacard.framework.ISOException used very often in applet development

javacard.framework

• public class AID

Encapsulates constants related to ISO7816-5

Mainly used at beginning of "communication" between applets

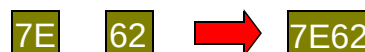
• public class Util

- Functions for byte array manipulation and comparison
 - **arrayCompare()** compares 2 arrays (*in constant time*)
 - **arrayCopy()** copies 1 array into another *atomically*
 - **arrayCopyNonAtomic()** copies 1 array into another *non-atomically*
 - **arrayFillNonAtomic()** fills an array with a **byte** value *non-atomically*
- Functions for type conversion (short/byte)

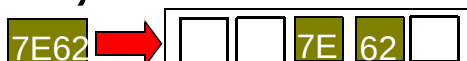
- **short getShort()**



- **short makeShort()**



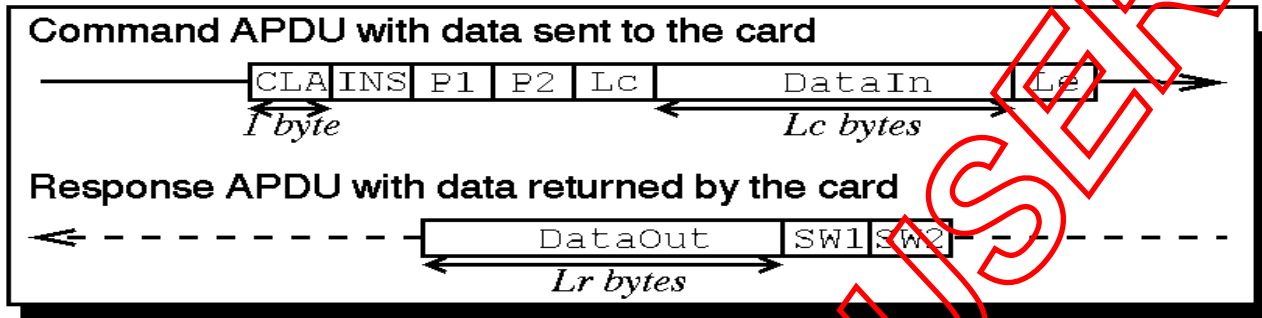
- **setShort()**



javacard.framework

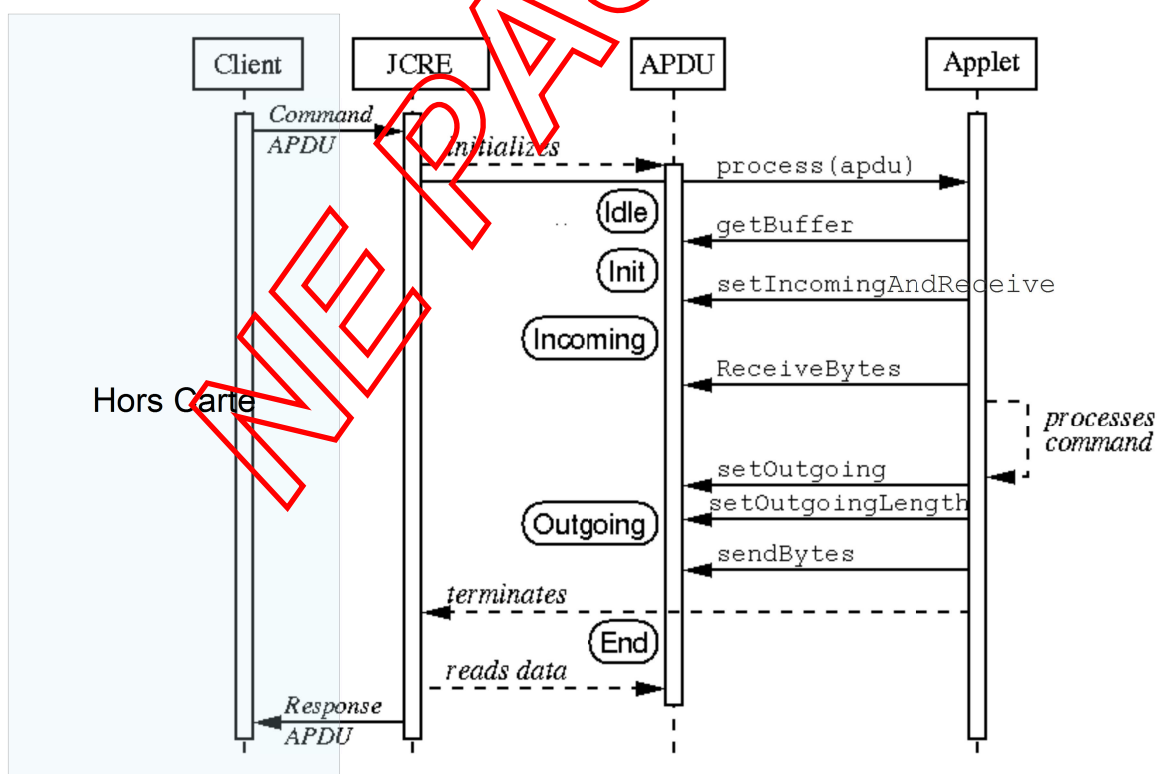
- **public final class APDU**

- Encapsulates 7816-4 APDU elements in an I/O buffer
- APDUs (Application Programming Data Units) define the format of the «data packets» exchanged between a reader and a card:
 - APDUs are defined for both commands and responses to hide the low level details of the protocol used (T=0 ou T=1)
 - Status words SW1 et SW2 are standardized (OK = 0x9000)



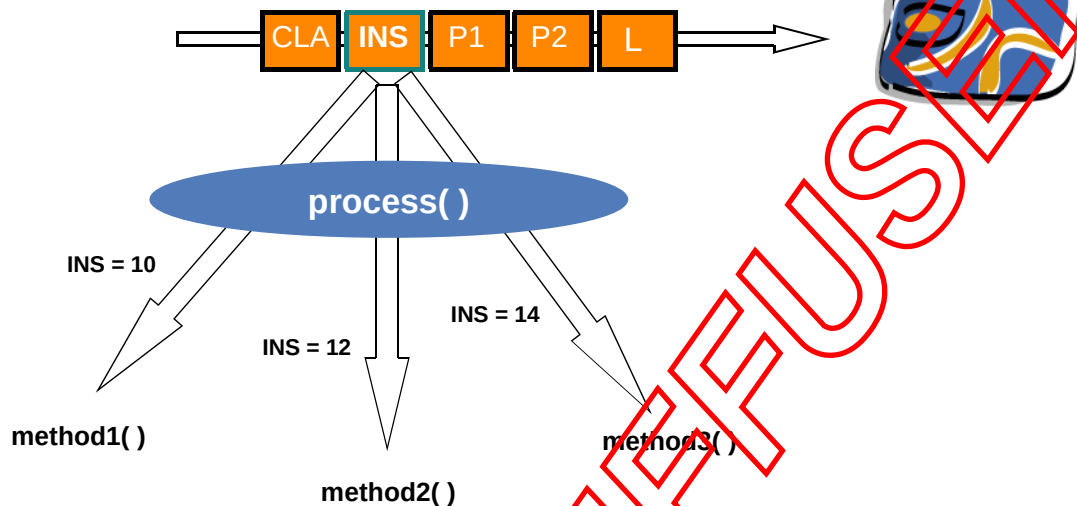
- Provides methods for **receiving** data and **sending** data
- Designed to be protocol independent
- The incoming / outgoing APDU data size may be bigger than the APDU buffer size and need to be read/write in portions by the applet.
- *APDU objects are owned by the JCRE (Temporary JCRE entry point objects)*

Handling the **APDU** class



Handling the **APDU** class

- `process(APDU apdu)` method is called
- `apdu.getBuffer()`
- CLA is checked and then INS is examined



javacard.framework

- **public interface ISO7816**

Contains only static constants

Public final static field with `SW_` prefixes define status words (standardized status response of ISO7816-4)

- e.g.: `SW_WRONG_P1P2` // Incorrect parameters (P1,P2) = 0x6B00

Fields with `OFFSET_` prefixes define constants/offset to use as index in the APDU buffer (byte array)

- e.g.: `OFFSET_CLA` // APDU header offset: CLA = 0

Fields with `_00` suffix require the low order byte to be customized appropriately

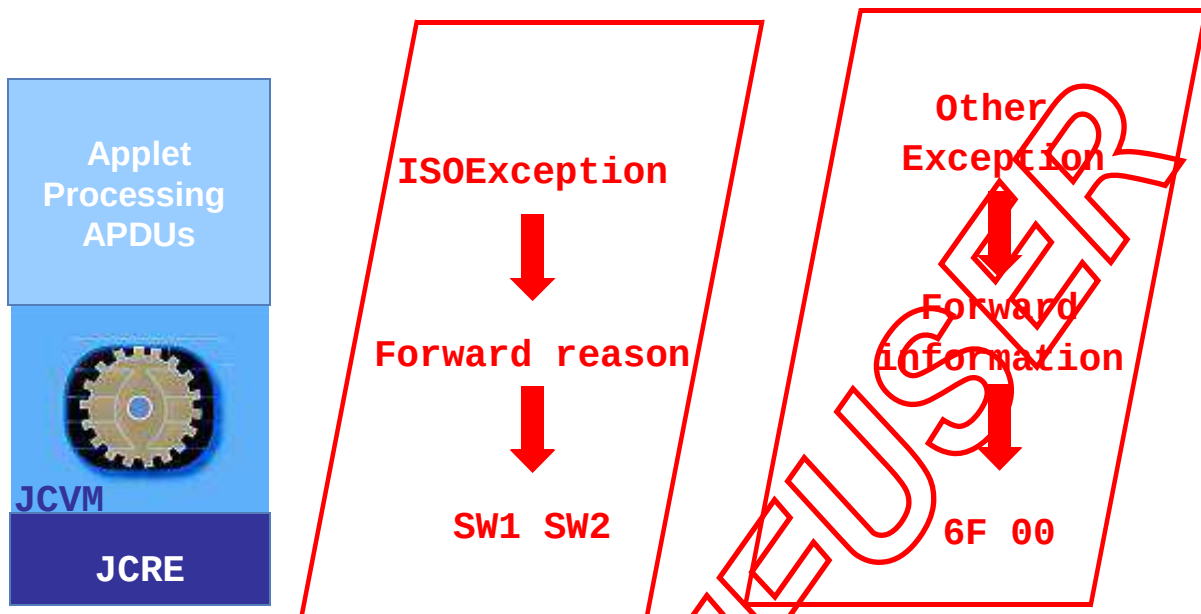
- e.g.: `SW_BYTES_REMAINING_00` // Response bytes remaining = 0x6100

- **public class ISOException**

Encapsulates an ISO 7816-4 response status word as its reason code

`throwIt(short reason)` allows to output a status word error

Java Card Exceptions



javacard.framework

• public interface PIN

- Define a PIN behavior and a PIN value
- Try limit maximum trials of an incorrect PIN before being blocked,
- Max PIN size, the maximum length of PIN,
- Try counter, the remaining number of trial,
- Validated flag => true if a valid PIN has been presented. Flag is reset on every card reset.
- If a transaction is in progress **update of the try counter shall not participate to the transaction,**

javacard.framework

- **public class OwnerPIN**

- Implements interface PIN
- Full rights for PIN code management
 - Update / Read capabilities -> for the code "owner" only
- Features:
 - direct modification of the validation flag
 - know whether or not the code has been validated
 - securely implemented against attacks

Methods available:

- check()
- isValidated()
- resetAndUnblock()
- setValidatedFlag()
- reset()
- getTriesRemaining()
- getValidatedFlag()
- update()

javacard.framework

public final class JCSystem

- Static methods (natives) for interaction with the JCRE
- Transactions
 - Only one level of transaction allowed
 - **beginTransaction(), commitTransaction(), abortTransaction()**
 - Limited by the RAM and EEPROM capacity
- Transient array
 - **makeTransientXXXArray(lenght, event)**
 - Build a transient array reinitialized at reset or deselection
 - **XXX** being Boolean, Byte, Short or Object
 - Event is **CLEAR_ON_DESELECT** or **CLEAR_ON_RESET**
- Object sharing
 - Build an object that inherit from **shareable**
 - Provide a reference with **getAppletShareableInterfaceObject (AID, Parameter)**

javacard.framework

- **public abstract class Applet**
 - Super class of all Java Card Applet, user applets must subclass Applet class.
 - Methods called by the JCRE
 - User applets MUST implements
 - **static void install(byte[] buf, short off, byte len)**
 - Create a new applet instance
 - **abstract void process(APDU apdu)**
 - Called for requesting the applet instance to execute a received command
 - User applets MAY overrides
 - **boolean select()**
 - Called when the applet instance is selected
 - **void deselect()**
 - Called when the applet instance is selected
 - Note that:
 - **protected void final register()**
 - Called during installation for registering the newly created instance

Développement Java Card

Construction d'applications carte

- Une application carte
 - Code de la carte (application serveur = applet Java Card)
 - Code dans le terminal (application cliente)
- Construction d'une application Java Card
 - Construction de l'application serveur (applet)
 - Implémentation des services
 - Installation de l'applet dans les cartes
 - Initialisation des services
 - Construction de l'application cliente
 - Invocation des services
- Définir le but de l'application puis spécifier le protocole de communication entre l'application cliente et l'applet Java Card.
 - Commandes à traiter
 - Réponses possibles
 - Cas d'erreurs

67

Construction d'une applet Java Card

- Rôle d'une applet
 - Maintenir son propre état : gestion des champs de l'applet, créer des objets et les référencer pour travailler
 - Répondre à des APDUs de commande (méthode `process()`) et retourner des APDUs de réponse
- Conception
 - Définir les structures de données nécessaires
 - Créer les objets de base à l'installation, initialiser les champs
 - Implémentation de la méthode `install()`
 - Définir les APDUs traités par la méthode `process()`
 - Implémentation d'un analyseur de commandes
 - Utilisation des champs et objets de l'applet
 - Utilisation des APIs
 - Définir les traitements à la sélection et à la désélection

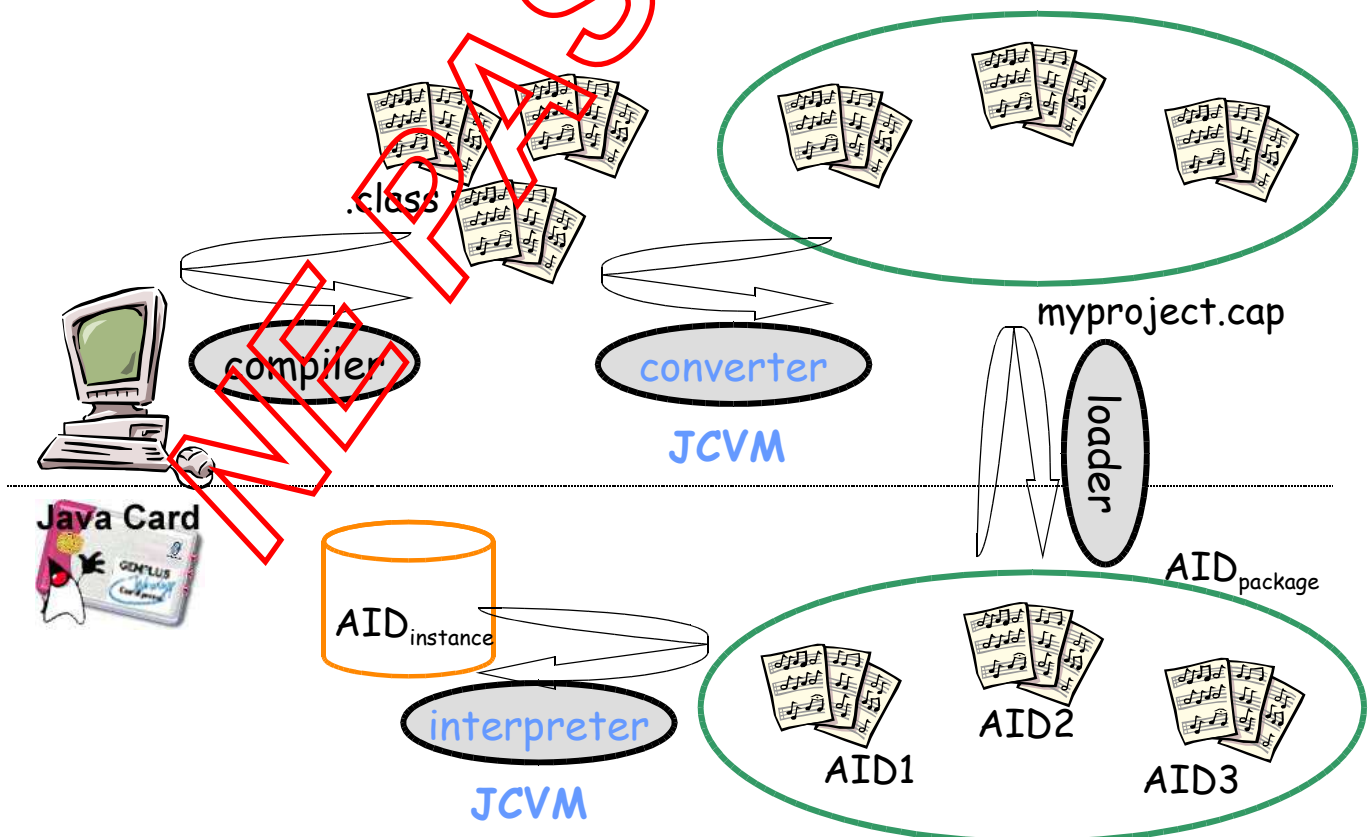
68

Construction d'une applet Java Card

- Fabrication (avec un IDE)
 - Compilation
 - Production de fichier class
 - Conversion
 - Production du fichier CAP
- Initialisation
 - Chargement du fichier CAP
 - Installation de l'applet (instanciation) via un appel à la méthode `install()` en lui donnant un AID
- Utilisation
 - Via l'AID défini pour cette applet

69

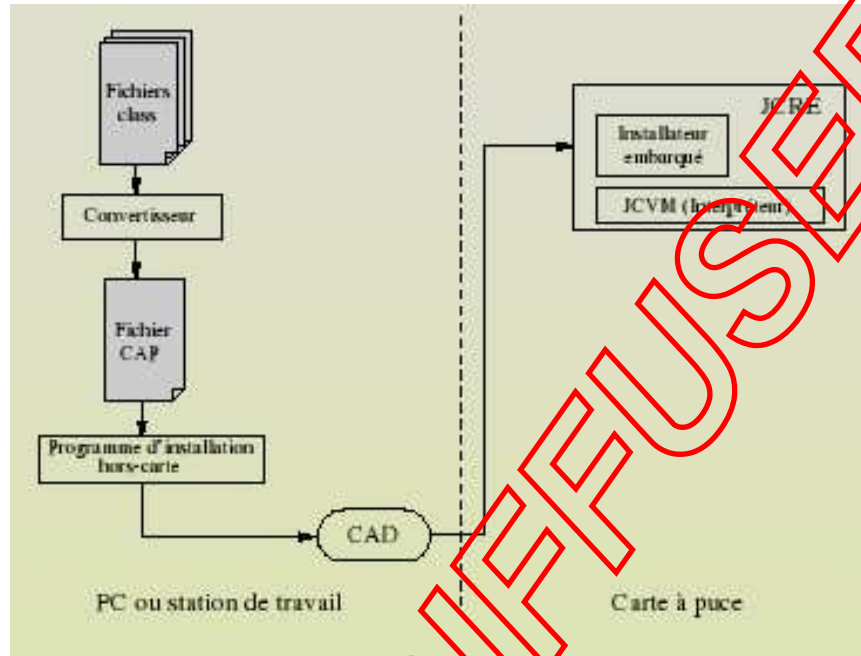
Applet development



70

Installateur Java Card

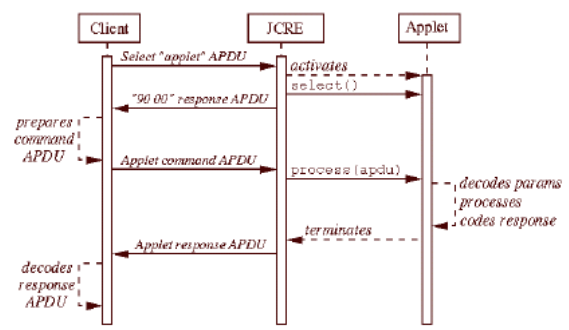
- L'installation se fait grâce à deux entités :
 - l'installateur Java Card et un programme d'installation hors carte
 - Suit quasiment tout le temps le standard GlobalPlatform



71

Construction de l'application cliente

- Rôle de l'application cliente
 - Stimuler l'application serveur et utiliser les résultats pour faire son travail (exemple informer le commerçant que oui l'ordre a bien été accepté)
- Conception
 - Définir les structures de données propre à l'application
 - Choisir son langage, un compilateur, éventuellement un framework développement (.NET, Java, etc.)
 - Choisir un framework de communication : PC/SC, JPC/SC, OCF, javax.smartcardio, etc.
 - Implémentation en utilisant ce framework du protocole de communication avec le serveur (carte)
 - Établissement de la liaison avec le serveur : envoie d'un APDU de sélection avec l'AID de l'applet
 - Invocation des services de l'applet :
 - Codage et envoie d'APDUs de commande conformes à ceux traités par l'applet
 - Réception et décodage des APDUs de réponse retournés par l'applet
- Fabrication
 - Avec l'IDE choisi
- Déploiement
 - sur les machines où le service sera utilisé
- Utilisation



Résumé construction d'applications

- Développement de l'applet carte
 - Utilisation de l'API Java Card
 - Essentiellement pour traiter des APDUs
 - Conversion et installation dans la carte
 - Format du bytecode et procédure d'installation dans la carte non «encore» définis par Java Card
 - Exécution par le JCRE
 - Sélection d'applet définie par Java Card
 - Réception et transmission d'APDUs par l'applet définies par Java Card
- Développement de l'application cliente
 - Utilisation d'une API de communication carte permettant de transporter des APDUs
 - Exemple : API OCF pour Java standard
 - Sélection de l'applet par son AID
 - Commande définie par Java Card
 - Échanges d'APDUs avec l'applet carte
 - Codage d'APDUs de commande et décodage d'APDUs de réponse conformes aux APDUs⁷³ traités par l'applet

Les difficultés du développement

- Client et applet communiquent par APDUs
 - Structure de données pauvre (tableau d'octets), peu explicite (pas de typage), et difficile à manipuler
 - Oblige à définir le contenu et la sémantique des messages échangés : décodage et encodage d'APDUs par le client et l'applet
- La spécification des échanges entre client et applet correspond à un protocole
 - Le format des messages APDUs échangés est utilisé comme spécification commune
 - Travail sur un protocole plutôt que sur des fonctionnalités
 - Nécessite une formation carte
- L'applet et le client implémentent un protocole
 - Code «dupliqué» dans les programmes clients et applets
 - Dans la carte : toutes les applets décodent des APDUs
 - Code «sensible»
 - Difficile à programmer : prévoir tous les cas, pas d'outils
 - Source d'erreurs

Exemple «Compteur» : Conception

- État de l'applet :
 - Maintient une valeur entière positive ou nulle (pas d'objets)
- Installation : création de l'objet Applet
 - Initialisation de la valeur du compteur
 - Enregistrement auprès du JCRE
- Opérations :
 - Lecture : retourne la valeur du compteur
 - Incrémentation/décrémentation : ajoute/soustrait un montant au compteur et retourne la nouvelle valeur du compteur
- Sélection et désélection :
 - Aucun traitement

75

Exemple «Compteur» : Conception

- APDUs traités par l'applet :

- int lire()

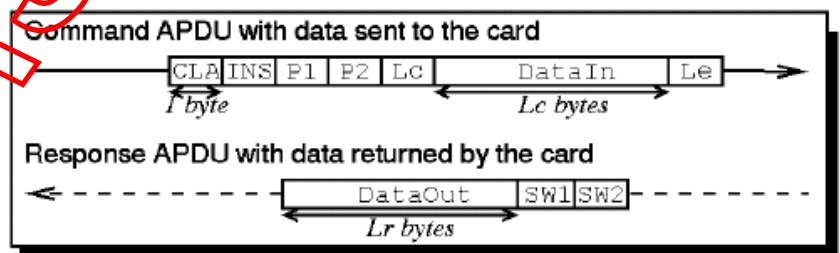
- Commande : A0 10 XX XX 00 04
- Réponse : RV3 RV2 RV1 RV0 90 00

- int incrementer(int)

- Commande : A0 12 XX XX 04 AM3 AM2 AM1 AM0 04
- Réponse : RV3 RV2 RV1 RV0 90 00

- int decrements(int) :

- Commande : A0 14 XX XX 04 AM3 AM2 AM1 AM0 04
- Réponse : RV3 RV2 RV1 RV0 90 00



76

Applet «Compteur» : Classe Applet

```
package fr.cryptis.compteur ;
import javacard.framework.* ;

public class Compteur extends Applet {

    private int valeur;

    public Compteur() { valeur = 0; }

    public static void install( byte[] bArray, short bOffset, byte bLength)
    { new Compteur().register(); }

    public void process( APDU apdu ) {
        byte[] buffer = apdu.getBuffer();
        if ( buffer[ISO7816.OFFSET_CLA] != (byte)0x10 )
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
        switch ( buffer[ISO7816.OFFSET_INS] ) {
            case (byte)0x10: ... // Opération de lecture
            case (byte)0x12: ... // Opération d'incrément
            case (byte)0x14: ... // Opération de décrémentation
            default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
        }
    }
}
```

77

Applet «Compteur» : décrémentation

```
case 0x14: // Opération de décrémentation
{
    // Réception des données
    byte octetsLus = apdu.setIncomingAndReceive();
    if ( octetsLus != 4 )
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    int montant = (buffer[ISO7816.OFFSET_CDATA]<<24) |
        (buffer[ISO7816.OFFSET_CDATA+1]<<16) |
        (buffer[ISO7816.OFFSET_CDATA+2]<<8) |
        buffer[ISO7816.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;
}
```

78

Rappel : Applet Java Card

- Une applet carte est un programme serveur de la Java Card
 - APDU de sélection depuis le terminal
 - Sélection par AID (chaque applet doit avoir un AID unique)
- Une fois installée dans la carte, l'applet est toujours disponible
 - Classe qui hérite de `javacard.framework.Applet`
 - Doit implémenter les méthodes qui interagissent avec le JCRE :
 - `public static void install( byte[] bArray, short bOffset, byte bLength )`
 - Appelée (une fois) par le JCRE quand l'applet est chargée dans la carte
 - Doit s'enregistrer auprès du JCRE (méthode `register()`)
 - `public boolean select()`
 - Appelée par le JCRE quand un APDU de sélection est reçu et désigne cette applet
 - Rend l'applet active
 - `public abstract void process( APDU apdu )`
 - Appelée par le JCRE quand un APDU de commande est reçu pour cette applet (doit être active)
 - `public void deselect()`
 - Appelée par le JCRE pour désélectionner l'applet courante

79

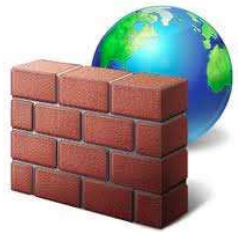
Les mécanismes de sécurité Java Card

Les composants de sécurité

- Les règles Java doivent être vérifiées. Aucun code hostile ne peut entrer dans la VM : **Byte Code Verification**
- Le modèle du bac à sable (Sand box): vous accédez seulement à ce qui vous appartient : **Firewall**
- Que les actions soient réalisées ou non, l'état de la VM et ceux des applications doivent rester cohérent : **Transaction**

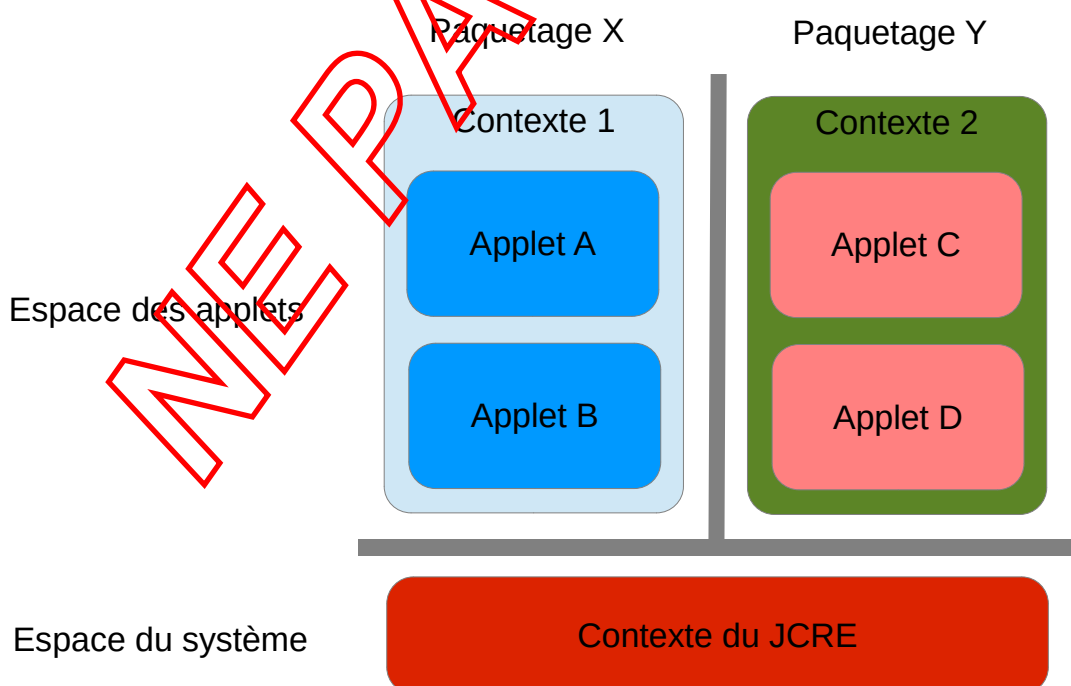
Firewall

Need of a firewall



- Add a security level, by providing **a protection against developer mistakes and design oversights** that may allowed sensitive data to leaked to another applet,
- Maintain a **strong segregation** level **between applets designed by different manufacturers**,
- If there is a need, **a mechanism to share information** has been defined in the specification,
- **All the checks are based on the notion of security context.**

La notion de contexte



Object ownership

- At any time, there is **only one active context within the virtual machine**: either the JCRE context or an applet's group context.
- When a new object is created, it is assigned an owning context: the currently active context.

JCRE CTX

b7

Current CTX

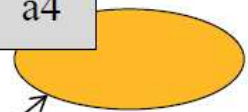
ToS



a4
12
b7

VM

a4



Object ownership

- Every object is owned by an applet instance which is identified by its AID.
- When executing in an instance method of an object the object's owner must be the currently active context,
- If the contexts do not match, the access is denied, and the comparison results in a Security Exception.

JCRE CTX

b7

Current CTX

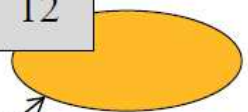
ToS

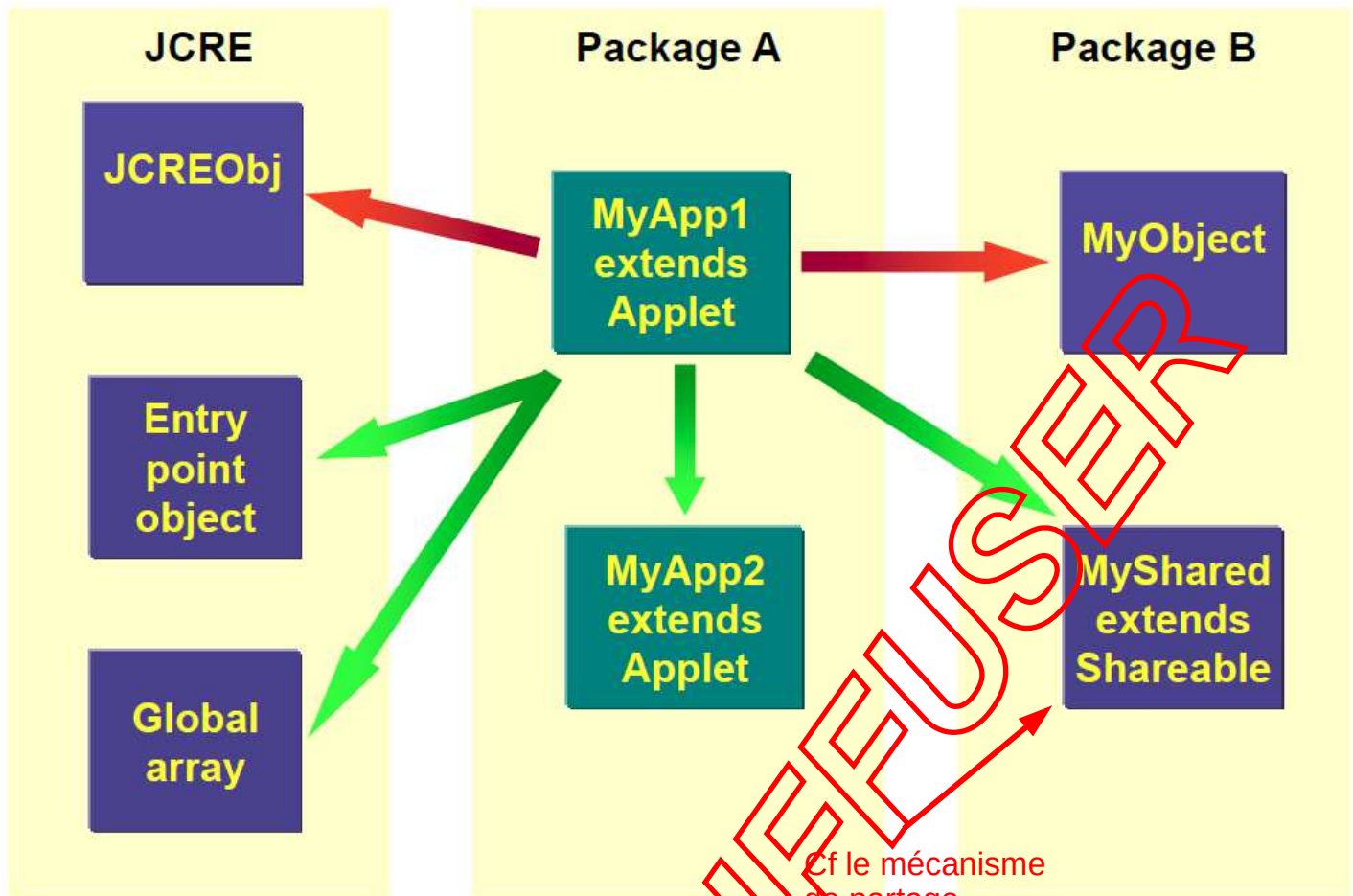


a4
12
b7

VM

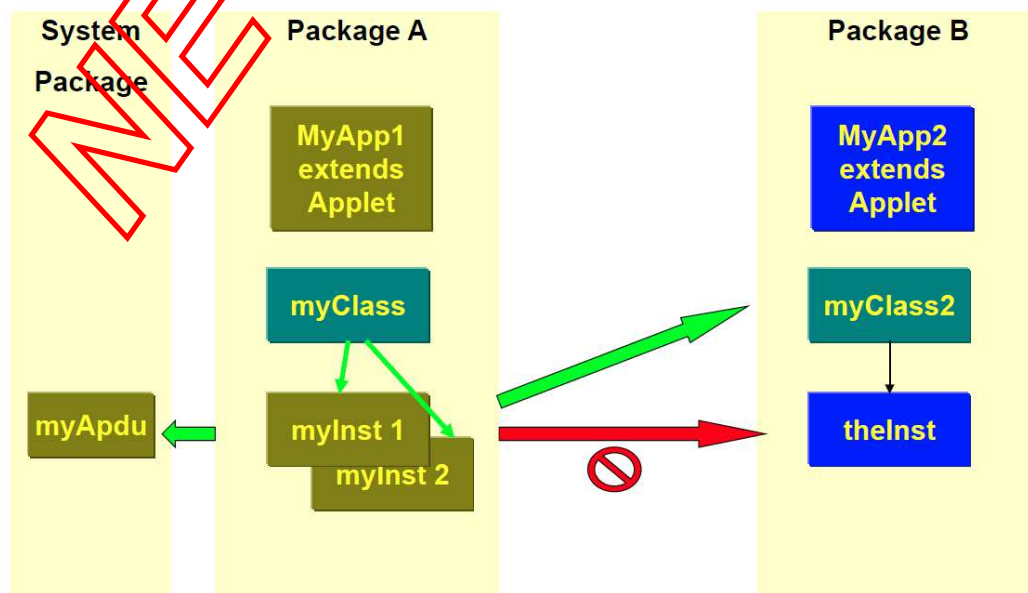
12





Static Fields and Methods

- Only instances of classes i.e. objects, are owned by applets; classes themselves are not.
- Static fields and methods are accessible from any applet context in the defining package (i.e. group context)
 - **Except they are public**: in this case, any applet context can access them.



Object Access across Context

- Sharing mechanisms are accomplished by the following means:

- JCRE privileges
- JCRE entry point objects
- Global arrays
- Shareable interfaces

Context Switch

- If Context 1 is the currently active context, and a method *m1* in an object owned by applet A is invoked, no context switch occurs. If method *m1* invokes a method *m2* in an object owned by applet B, again no context switch occurs (in spite of the object “owner” change), and no firewall restrictions apply.
- However, if the method *m2* now calls a method *m0* in an object owned by applet C, firewall restrictions apply and, if access is allowed, a context switch shall occur. Upon return to method *m2* from the method *m0*, the context of applet B is restored.

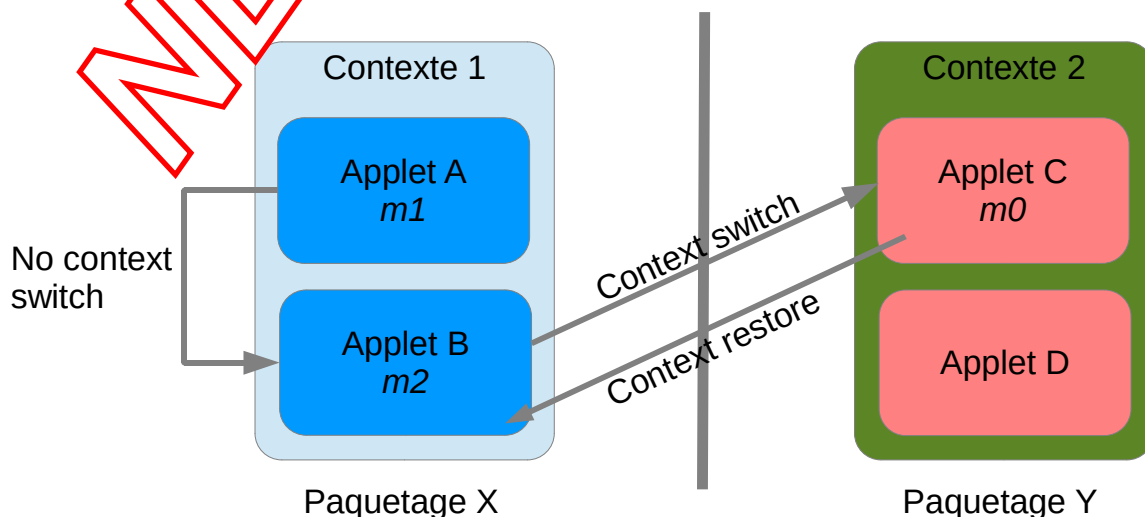


Illustration du fonctionnement du firewall

```
package Z;
public class Foo {
    public static Bar bar;
}

public class Bar {
    public void test(){ ... }
}
```

```
package X;
import Z.*;

public class A extends Applet {
    public void process(APDU apdu) {
        ...
        Foo.bar = new Bar();
        ...
    }
}
```

```
package Y;
import Z.*;

public class C extends Applet {
    public void process(APDU apdu) {
        ...
        Bar bar = Foo.bar;
        bar.test();
        ...
    }
}
```

Applet isolation

- The firewall provides a strong isolation mechanism between contexts
- Shareable interfaces are a feature in the API to enable applet interaction through a Shareable Interface Object (SIO),
- From the owning context, the SIO is a normal object whose fields and methods can be accessed,
- To any other context, **only the methods defined in the shareable interface are accessible**,
- All other fields and methods are protected by the firewall.

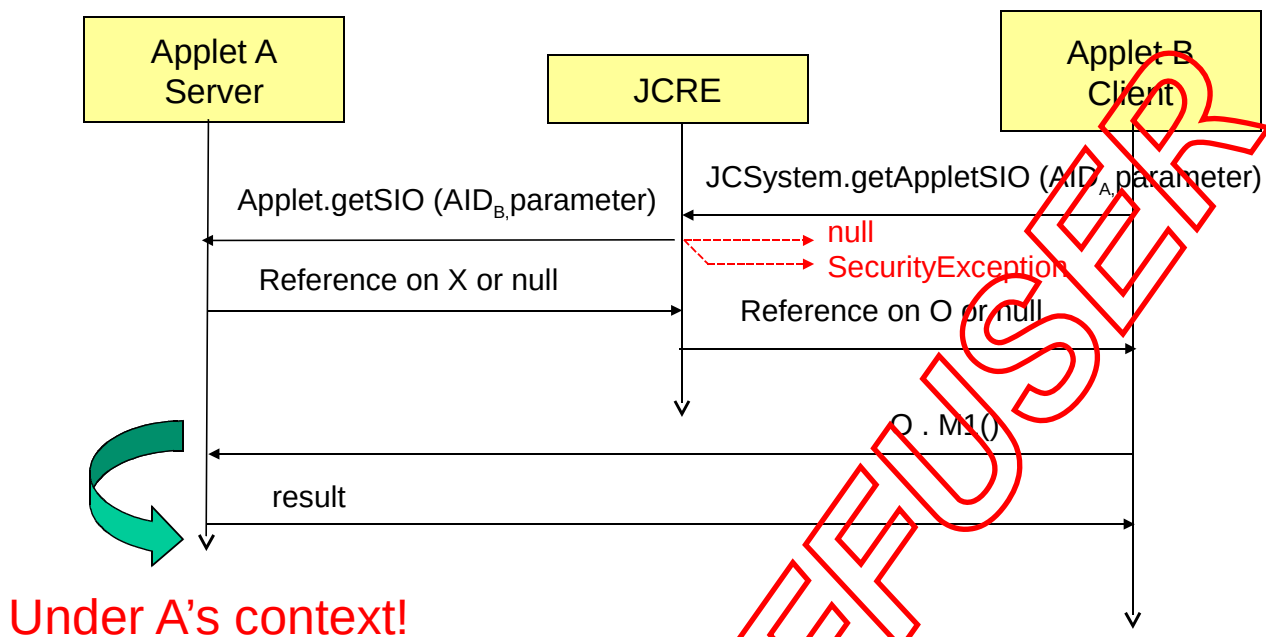
Sharing mechanism

- Methods of an object that implements a shareable interface (SI) can be invoked through the firewall
- **Once shared an object (SIO) cannot be un-shared**
 - In addition **it can be given to an untrusted party**
- However the caller needs to obtain a reference to this object
 - Applet *A* agree to share with applet *B*
 - Applet *A* declares to implement a shareable interface and implements the services
 - Applet *B* access the services by obtaining an object reference and invoking the service methods.

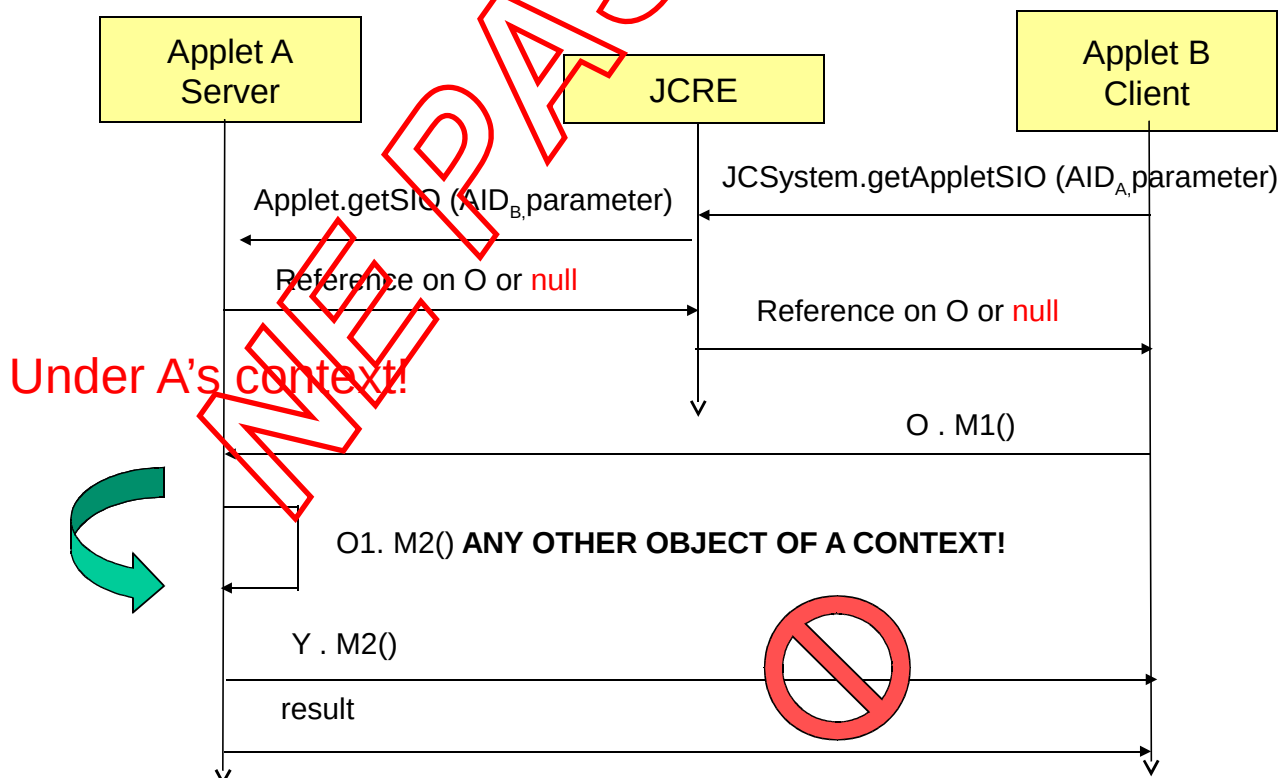
Server Applet A

- *A* defines a Shareable Interface (SI) that extends the interface `javacard.framework.Shareable`
- Defines a class *C* that implements SI with the methods defined in SI,
- Defines other methods protected by the firewall
- *A* creates an object instance of class *C*

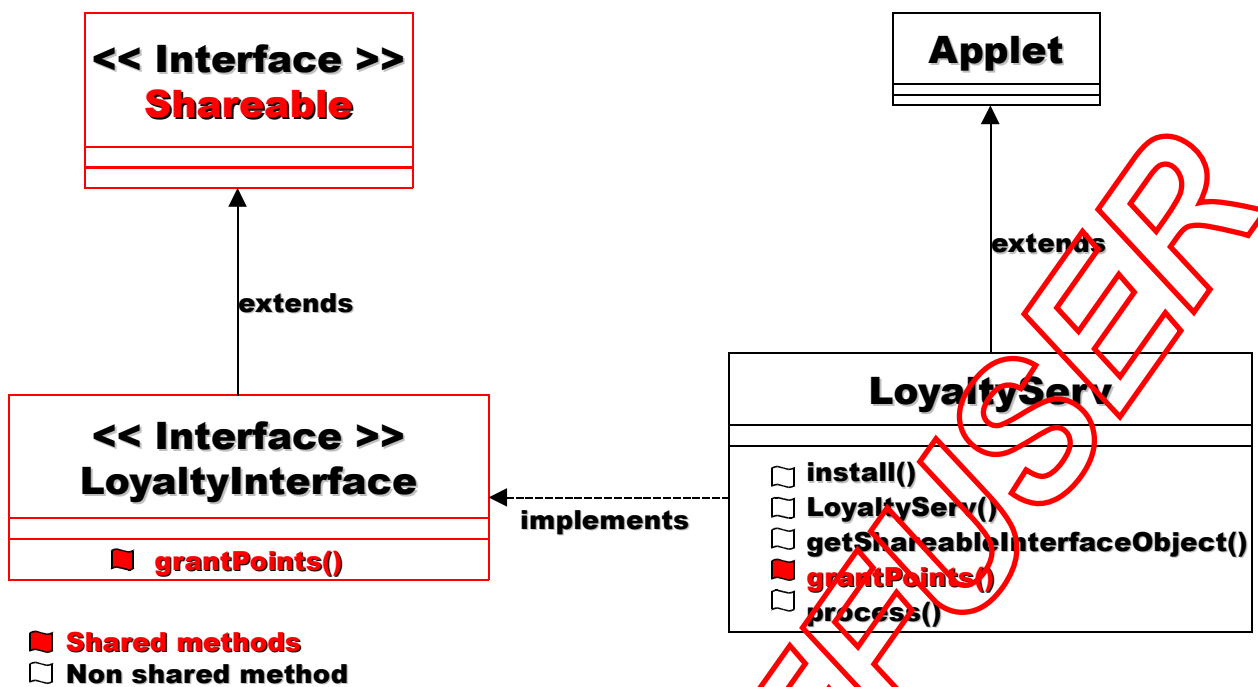
Shareable Interfaces



Shareable Interfaces



Example



Example server side

The server must create a shareable interface that extends `javacard.framework.Shareable`

```

package myPackage
import javacard.framework.Shareable
public interface LoyaltyInterface extends Shareable {
    public void grantPoint (short amount)
}
    
```

- The server implements the interface

```

package myPackage
public class LoyaltyServ extends Applet implements LoyaltyInterface {
    private short miles;
    public void grantPoint (short amount) {
        miles = (short) (miles+amount);
    }
}
    
```

Example client side

- The client needs to know the server AID,

```
import myPackage.LoyaltyInterface;
public class LoyaltyClient extends Applet {
...
// request SIO from the server
LoyaltyInterface sio = (LoyaltyInterface)
    JCSysyem.getAppletShareableInterfaceObject(AIDServer, (byte) 0)
if (sio==null){ISOException.throwIt((short)0x6999);}
sio.grantPoint(theAmount);
```

- The JCRE looks up the server applet and invokes the server `getShareableInterfaceObject` with the client AID as parameter !!!

Example server side

- The server implements the interface

```
package myPackage;
public class LoyaltyServ extends Applet implements LoyaltyInterface{
    private short miles;
    public void grantPoint (short amount) {
        miles = (short) (miles+amount);
    }
    public Shareable getShareableInterfaceObject (AID clientAID,
        byte param){
        if (clientAID.equals(myFriend)){
            return ((Shareable)this)}
        else {return null}
    }
...
}
```

Example client side

- The client needs to know the server AID,

```
import myPackage.LoyaltyInterface;
public class LoyaltyClient extends Applet {
...
    // request SIO from the server
    LoyaltyInterface sio = (LoyaltyInterface)
        JCSysSystem.getAppletShareableInterfaceObject (AIDServer, (byte)
        0);
    if (sio == null){ISOException.throwIt((short)0x6999);}
    sio.grantPoint(theAmount);
...
}
```

Type confusion with sharing

- Poll *et al.* succeeded with some old cards with type confusion
- They have been able to recover twice the size of the array using applets compiled and loaded separately,

- Modify the signature and try to perform an array overflow,

```
public interface MyIntextends Shareable {
    void accArray(short[] array);}
```

with for the client

```
public interface MyIntextends Shareable {
    void accArray(byte[] array);}
```

- The server will treat the byte array as a short array.

- However there is still the firewall... thus the trick is `byte[] giveArray();`

- They demonstrate that some cards allow it.

- References :

- Wojciech Mostowski and Erik Poll. Malicious Code on Java Card Smartcards: Attacks and Countermeasures. Proceedings, Smart Card Research and Advanced Application Conference CARDIS 2008, Egham, U.K., September 2008. LNCS 5189, pages 1-16,

Attention

- Protection :
 - Faire une authentification mutuelle entre les applets
 - Insuffisant s'il y a collusion entre un partenaire et une application qui n'est pas de confiance (ils peuvent s'échanger les clés)
 - Il faut se fier à la plate-forme avec les méthodes suivantes.
 - Utiliser `JCSystem.getAID()` et `JCSystem.getPreviousContextAID()` pour empêcher des accès non souhaités !

APIs Cryptographiques

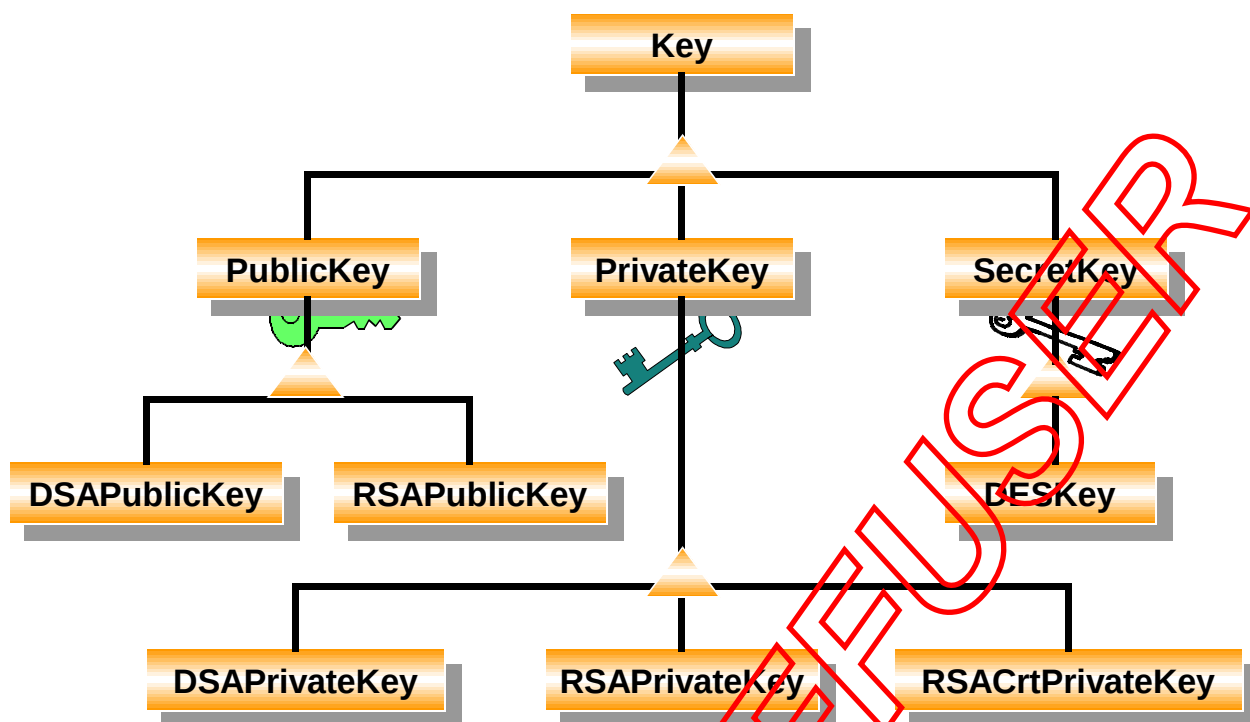
javacard.security

- Contains interfaces for managing keys
 - Key, SecretKey, DESKey, PublicKey, RSAPublicKey, DSAPublicKey, PrivateKey, RSAPrivateKey, DSAPrivateKey, RSAPrivateCrtKey
- Contains objects for realizing cryptographic operations
 - KeyBuilder, Signature, MessageDigest, RandomData, CryptoException

3.0 Classic Edition API

- AES: Advanced Encryption Standard (FIPS-197)
- SEED Algorithm Specification: KISA - Korea Information Security Agency Standard Names for Security and Crypto Packages
- SHA (SHA-1): Secure Hash Algorithm, as defined in Secure Hash Standard, NIST FIPS 180-1
- SHA-256, SHA-384, SHA-512: Secure Hash Algorithm, as defined in Secure Hash Standard, NIST FIPS 180-2
- MD5: The Message Digest algorithm RSA-MD5, as defined by RSA DSI in RFC 1321
- RIPEMD-160: as defined in ISO/IEC 10118-3:1998 Information technology – Security techniques - Hash-functions - Part 3: Dedicated hash-functions
- DSA: Digital Signature Algorithm, as defined in Digital Signature Standard, NIST FIPS 186
- DES: The Data Encryption Standard, as defined by NIST in FIPS 46-1 and 46-2
- RSA: The Rivest, Shamir and Adleman Asymmetric Cipher algorithm
- ECDSA: Elliptic Curve Digital Signature Algorithm
- ECDH: Elliptic Curve Diffie-Hellman algorithm
- AES: Advanced Encryption Standard (AES), as defined by NIST in FIPS 197
- HMAC: Keyed-Hashing for Message Authentication, as defined in RFC-2104

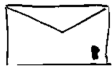
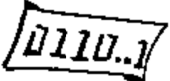
The key hierarchy in `javacard.security`



`javacard.security`

Interfaces Summary	
Key	<code>clearKey()</code> , <code>getSize()</code> , <code>getType()</code> , <code>isInitialized()</code> .
SecretKey extends Key DESKey extends SecretKey	<code>getKey()</code> , <code>setKey()</code> .
PrivateKey extends Key RSAPrivateKey extends PrivateKey RSAPrivateCrtKey extends PrivateKey	<code>getExponent()</code> , <code>getModulus()</code> , <code>setExponent ()</code> , <code>setModulus()</code> . <code>set[P,Q,PQDP1,DQ1] ()</code> , <code>get[P,Q,PQDP1,DQ1] ()</code> .
PublicKey extends Key RSAPublicKey extends PublicKey	<code>getExponent()</code> , <code>getModulus()</code> , <code>setExponent ()</code> , <code>setModulus()</code> .

javacard.security

javacard.security		
Class Summary		
KeyBuilder		Allows to create a key object factory.
Signature		Is the base class for signature algorithms .
RandomData	..8569765..	Generates random number .
MessageDigest		Is the base class for hashing algorithms .
KeyPair		Enable generation of a KeyPair

KeyBuilder

javacard.security	
KeyBuilder	
buildKey(type,length,encrypt)	Create a key with a specific type and length.
TYPE_DES TYPE_DES_TRANSIENT_DESELECT TYPE_DES_TRANSIENT_RESET TYPE_RSA_PUBLIC TYPE_RSA_PRIVATE TYPE_RSA_CRT_PRIVATE	
LENGTH_DES (64 bits) LENGTH_DES_2KEY (128 bits) LENGTH_DES_3KEY (192 bits) LENGTH_RSA_512 LENGTH_RSA_768 LENGTH_RSA_1024	

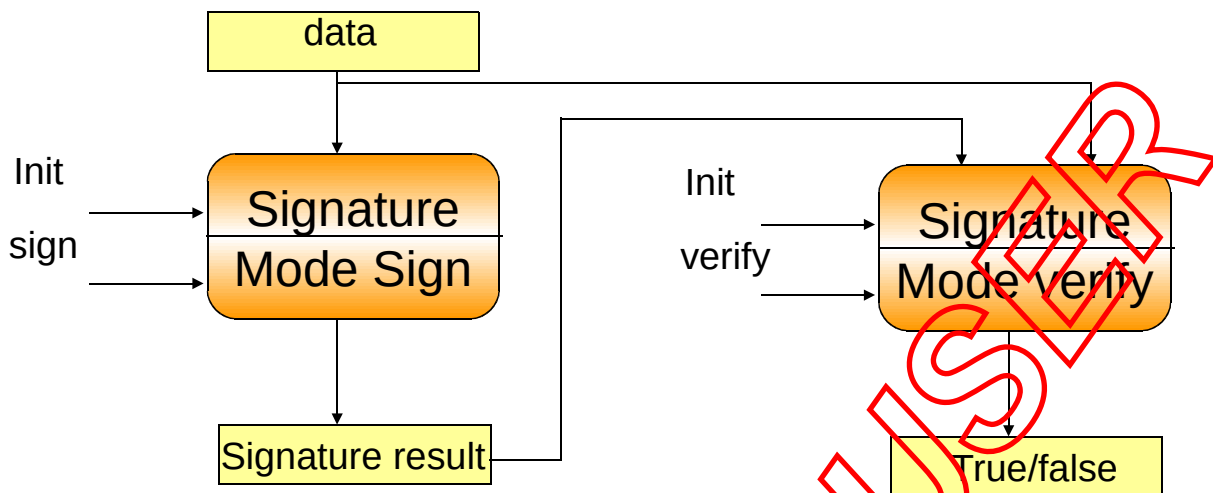
Signature

javacard.security	
Signature	methods
getInstance()	Creates a Signature object instance of the selected algorithm.
init()	Initializes the Signature object.
update()	Accumulates a signature of the input data.
sign()	Generates the signature of all/last input data.
verify()	Verifies the signature of all/last input data against the passed in signature.
getAlgorithm()	Gets the Signature algorithm.
getLength()	Returns the byte length of the signature.
fields	
ALG_DES_MAC4_NOPAD ALG_DES_MAC8_NOPAD ALG_RSA_SHA_PKCS1	
MODE_SIGN MODE_VERIFY	TheMode used in init() method.

javacard.security

- **KeyBuilder**
 - Creates non initialized cryptographic keys for signature and cipher algorithms
 - `buildKey` method create the key and returns a `Key` interface
- **Signature class**
 - Is an abstract class for signature algorithms
 - Implementations must extend this class and implement all the abstract methods,
 - Contains all signature constants value for `getInstance` algorithms parameters specification

Principe



Example of Signature for signature usage

- Once inside the applet:

- Create a **private** key object using `buildKey()` method

```
RSAPrivateKey privrsakey =  
(RSAPrivateKey)KeyBuilder.buildKey(KeyBuilder.TYPE_RSA_PRIVATE,  
KeyBuilder.LENGTH_RSA_512, false);
```

- Initialize your private key object using `setExponent()` and `setModulus()` methods

```
privrsakey.setExponent(buffer_exp, offset_exp, length_exp);  
privrsakey.setModulus(buffer_mod, offset_mod, length_mod);
```

- Create a signature object using `getInstance()` method

```
Signature sgn = Signature.getInstance(Signature.ALG_RSA_SHA_PKCS1, false);
```

- At each usage:

- Initialize your signature object using `init()` method

```
sgn.init(privrsakey, Signature.MODE_SIGN);
```

- Compute your signature

- feed the data using `update()` method
- compute signature using `sign()` method

```
sgn.update(a1, offset1, length1);
```

```
...
```

```
sgn.sign(aX, offsetX, lengthX, sgn_buffer, sgn_offset);
```

Example of Signature for verification usage

- Once inside the applet:

- Create a **public** key object using `buildKey()` method

```
RSAPublicKey pubrsaKey =  
(RSAPublicKey)KeyBuilder.buildKey(KeyBuilder.TYPE_RSA_PUBLIC,  
KeyBuilder.LENGTH_RSA_512, false);
```

- Initialize your public key object using `setExponent()` and `setModulus()` methods

```
pubrsaKey.setExponent(buffer_exp, offset_exp, length_exp);  
pubrsaKey.setModulus(buffer_mod, offset_mod, length_mod);
```

- Create a signature object using `getInstance()` method

```
Signature sgn = Signature.getInstance(ALG_RSA_SHA_PKCS1, false);
```

- At each usage:

- Initialize your signature object using `init()` method

```
sgn.init(pubrsaKey, Signature.MODE_VERIFY);
```

- Compute your signature

- feed the data using `update()` method
 - verify signature using `verify()` method

```
sgn.update(a1, offset1, length1);
```

```
...  
sgn.verify(aX, offsetX, lengthX, sig_buffer, sig_offset, sig_length);
```

Another example of Signature usage

- Once inside the applet:

```
DESKey k = (DESKey)KeyBuilder.buildKey(KeyBuilder.TYPE_DES,  
KeyBuilder.LENGTH_DES, false);
```

```
k.setKey(buffer, offset, length);
```

```
Signature s = Signature.getInstance(ALG_DES_MAC_NOPAD, false);
```

- At each usage:

```
s.init(k, Signature.MODE_SIGN);
```

```
s.update(in_buff, in_ofs, in_len);
```

```
s.sign(in_buff, in_ofs, in_len, out_buff, out_ofs);
```

```
s.init(k, Signature.MODE_VERIFY);
```

```
s.update(in_buff, in_ofs, in_len);
```

```
s.verify(in_buff, in_ofs, in_len, out_buff, out_ofs, out_len);
```

MessageDigest

javacard.security	
MessageDigest	
getInstance()	Creates a MessageDigest object instance of the selected algorithm.
update()	Accumulates a hash of the input data.
doFinal()	Generates a hash of all/last input data.
reset()	Resets the MessageDigest object to the initial state for further use.
getAlgorithm()	Gets the Message Digest algorithm.
getLength()	Returns the byte length of the hash.
ALG_SHA ALG_MD5	

Example of MessageDigest usage

- Once inside the applet:
 - Create a message digest object using `getInstance()` method

```
MessageDigest msgd = MessageDigest.getInstance(MessageDigest.ALG_SHA, false);
```
- At each usage:
 - Re-Initialize your message digest object using `reset()` method

```
msgd.reset();
```
 - Compute your digest
 - feed the data using `update()` method
 - compute the message digest using `doFinal()` method

```
msgd.update(a1, offset1, length1);  
...  
msgd.doFinal(aX, offsetX, lengthX, msgd_buffer, msgd_offset);
```

RandomData

javacard.security	
RandomData	
<code>getInstance()</code> <code>generateData()</code> <code>setSeed()</code>	Creates a RandomData instance of the selected algorithm. Generates random data in a buffer. Seeds the random data generator*.
<code>ALG_SECURE_RANDOM</code>	

**software part which modifies the random number given by the hardware part to enhance security.*

Example of RandomData usage

- Once inside the applet:
 - Create a random number object using `getInstance()` method
`RandomData rd = RandomData.getInstance(RandomData.ALG_SECURE_RANDOM);`
- At each usage:
 - If needed set a seed for your random number object using `setSeed()` method
`rd.setSeed(seed_buffer, seed_offset, seed_length);`
 - Generate random data using `generateData()` method
`rd.generateData(rd_buffer, rd_offset, rd_length);`

Example of KeyPair usage

- Once inside the applet:

- Create a Key pair object by instantiating the class

```
KeyPair kp = new KeyPair(KeyPair.ALG_RSA_CRT, (short) 1024);
```

- Generate the Key pair using `genKeyPair()` method

```
kp.genKeyPair();
```

- At each usage:

- Public and private keys of the pair can be get using `getPublic()` and `getPrivate()` methods

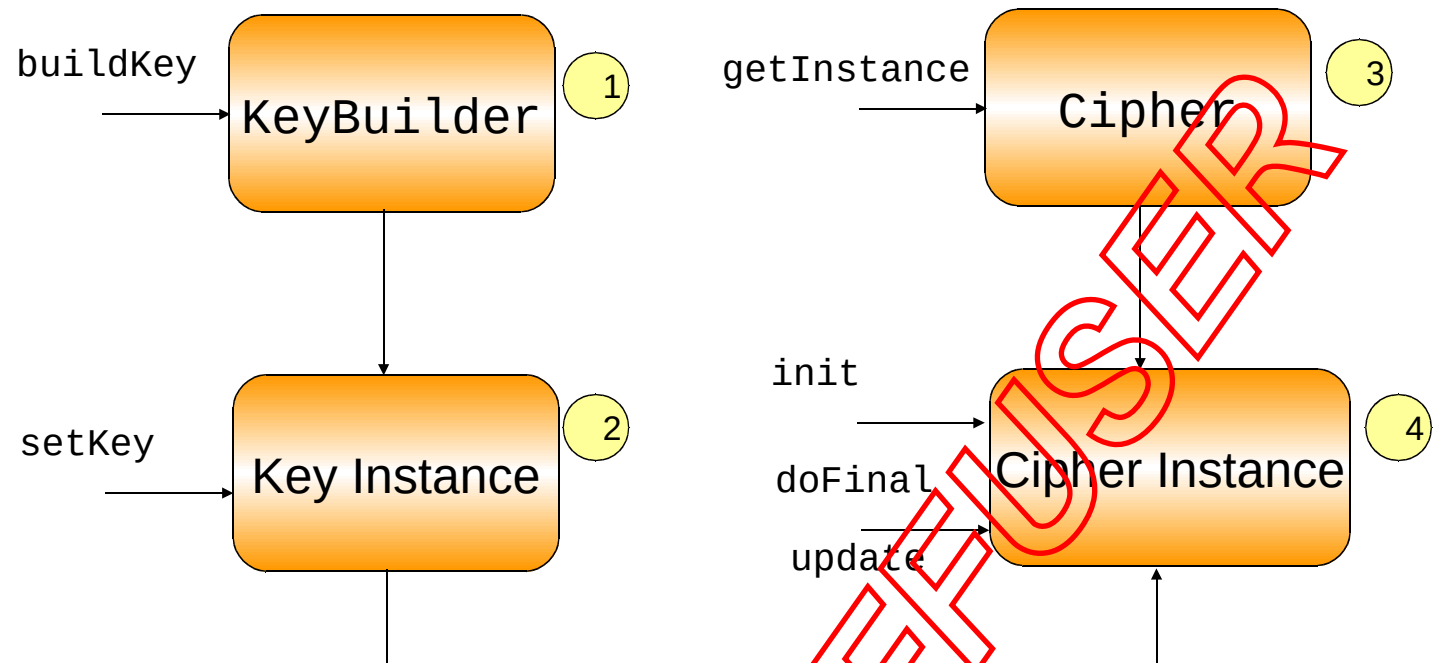
```
RSAPublicKey pub = (RSAPublicKey) kp.getPublic();
```

```
RSAPrivateCrtKey priv = (RSAPrivateCrtKey) kp.getPrivate();
```

javacardx.security.Cipher

javacardx.crypto	
Cipher	methods
getInstance()	Creates a Cipher object instance of the selected algorithm.
init()	Initializes the Cipher object with the appropriate Key and encrypt or decrypt mode.
update()	Generates encrypted/decrypted output from input data.
doFinal()	Generates encrypted/decrypted output from all/last input data.
getAlgorithm()	Gets the Cipher algorithm.
fields	
MODE_DECRYPT MODE_ENCRYPT	To specify the mode used in the <code>update()</code> or <code>doFinal()</code> method.
ALG_DES_CBC_NOPAD ALG_DES_ECB_NOPAD ALG_RSA_NOPAD	

Principe



Example of Cipher usage

- Once inside the applet:
 - Create a Cipher object using `getInstance()` method
`Cipher cipher = Cipher.getInstance(Cipher.ALG_RSA_NOPAD, false);`
- At each usage:
 - Initialize your cipher object using `init()` method
`cipher.init(pub, Cipher.MODE_ENCRYPT);` // pub is the RSAPublicKey
 - Compute your encrypted/decrypted data
 - feed the data using `update()` method
 - compute resulting data using `doFinal()` method`cipher.update(a1, offset1, length1);`
`...`
`cipher.doFinal(aX, offsetX, lengthX, cipher_buffer, cipher_offset);`

Another example of Cipher usage

- Once inside the applet:

```
DESKey k = (DESKey) KeyBuilder.buildKey(KeyBuilder.TYPE_DES,
    KeyBuilder.LENGTH_DES, false);
k.setKey(buffer, offset, length);
Cipher c = Cipher.getInstance(Cipher.ALG_DES_MAC_NOPAD,
    false);
```
- At each usage:

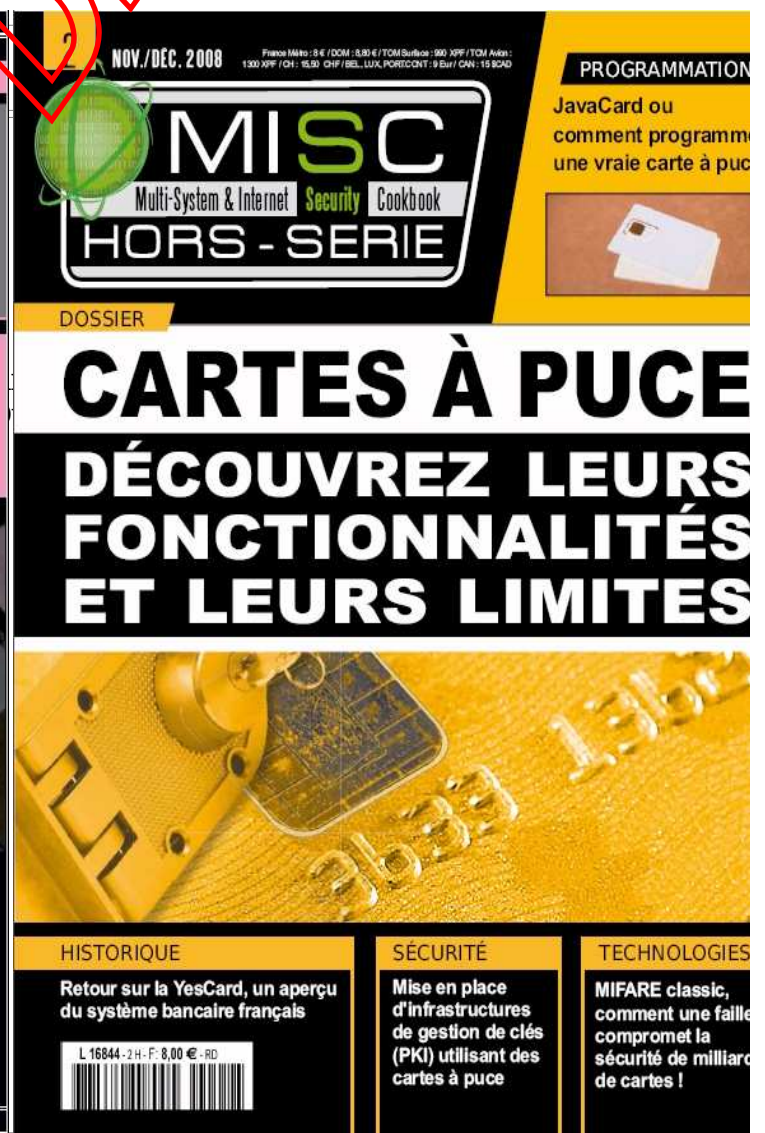
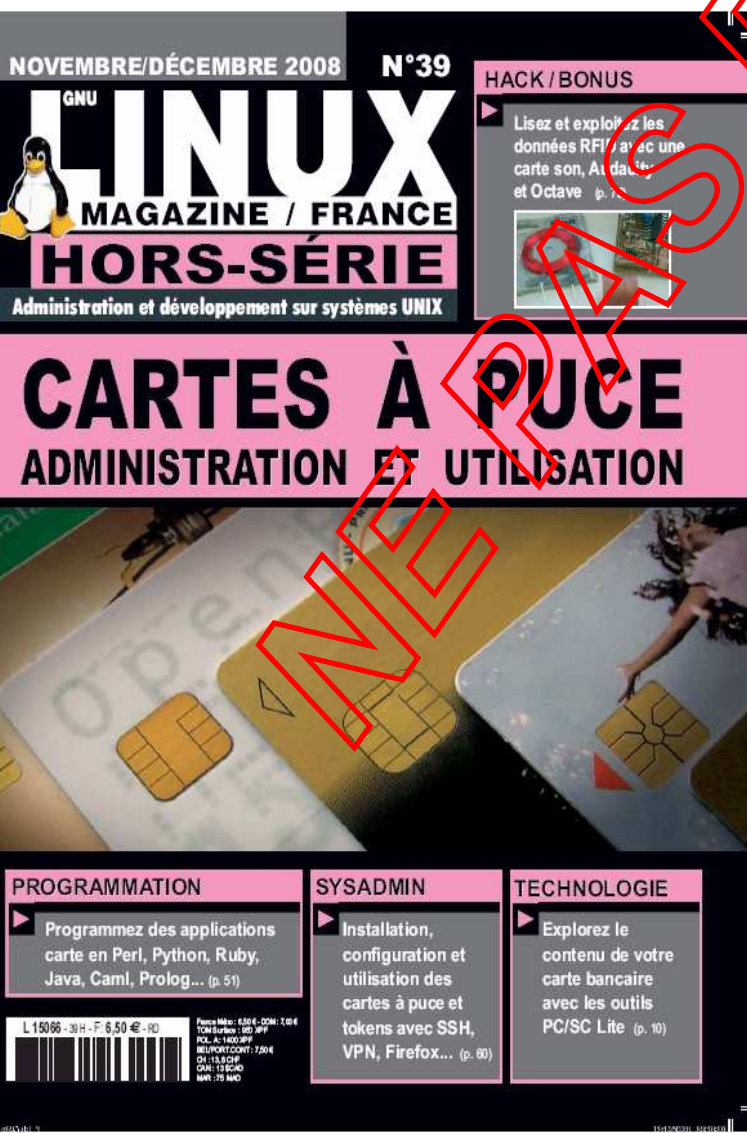
```
c.init(k, Cipher.MODE_ENCRYPT);
c.update(in_buff, in_ofs, in_len);
c.doFinal(in_buff, in_ofs, in_len, out_buff, out_ofs);
```

Hints

- `getInstance (algorithm, external access)` method
 - “external access” boolean parameter
 - `true`: access from other applets via the `Shareable` interface
 - `false`: access only from applets in the same package, if it is selected.
- `update()` method uses
 - temporary storage for intermediate results: used only if all input data cannot be contained in one byte array.

And some more...

- Package `javacardx.biometry`: Extension package that contains functionality for implementing a biometric framework on the Java Card platform.
 - `BioBuilder`, `BioException`, `BioTemplate`, `OwnerBioTemplate`, `SharedBioTemplate`
- Package `javacardx.framework.math`: Extension package that contains common utility functions for BCD math and parity computations.
 - `BCDUtil`, `BigNumber`, `ParityBit`
- `javacardx.apdu`: Extension package that enables support for ISO7816 specification defined optional APDU related mechanisms.
- `javacardx.tlv`: Extension package that contains functionality for managing storage for BER TLV formatted data, based on the ASN.1 BER encoding rules of ISO/IEC 8825-1:2002, as well as parsing and editing BER TLV formatted data in I/O buffers.



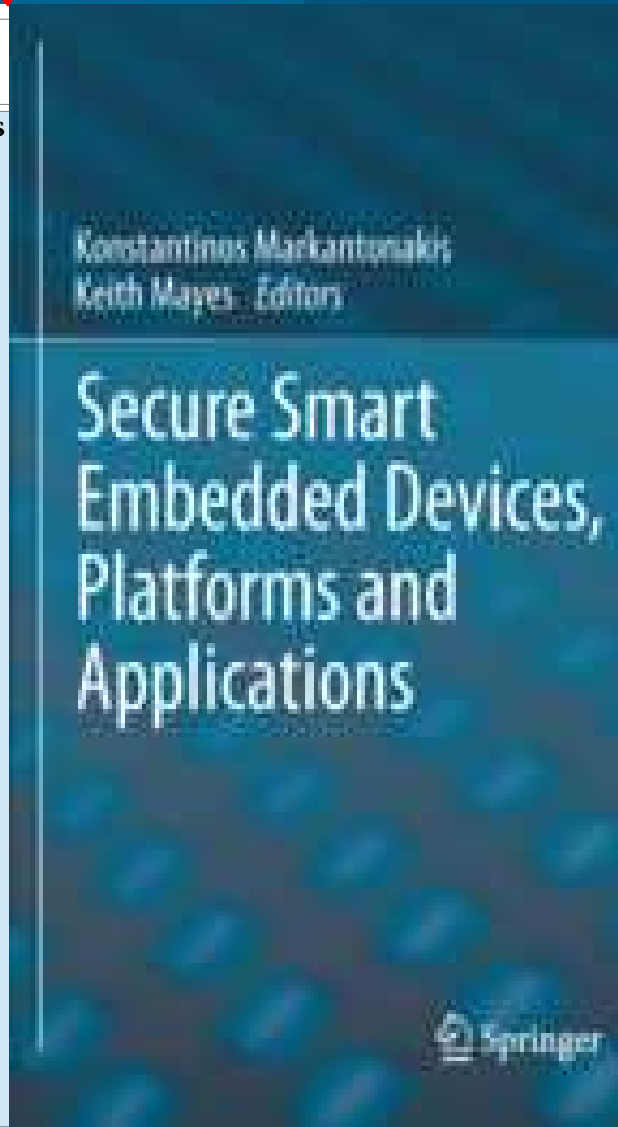
[Click to edit the title text format](#)

Smart Cards, Tokens, Security and Applications
Mayes, Keith; Markantonakis, Konstantinos (Eds.)
2008, XXXVIII, 392 p. 128 illus., Hardcover
ISBN: 978-0-387-72197-2



[Click to edit the title text format](#)

Secure Smart Embedded Devices, Platforms and Applications
Konstantinos Markantonakis, Keith Mayes
2014, XLI, 568 p. 135 illus., Hardcover
ISBN: 978-1-4614-7914-7



Les cartes à puce

BOUZEFRANE Samia, PARADINAS Pierre

Ouvrage 326 p. 15.6x23.4 cm Relié

Les cartes à puce

sous la direction de
Samia Bouzefrane
Pierre Paradinas

Hermès

Lavoisier

Questions ?

© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com



"My smart-card doesn't
understand me."

**Si votre carte ne vous comprend pas,
la comprenez vous mieux ?**

