Java Virtual Machine

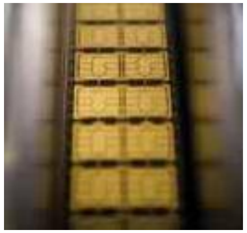
Carte à puce et Java Card

ATAC

2011-2012

Jean-Louis Lanet

Jean-louis.lanet@unilim.fr



Université de Limoges

SUFOP Service Universitaire de Formation Permanente

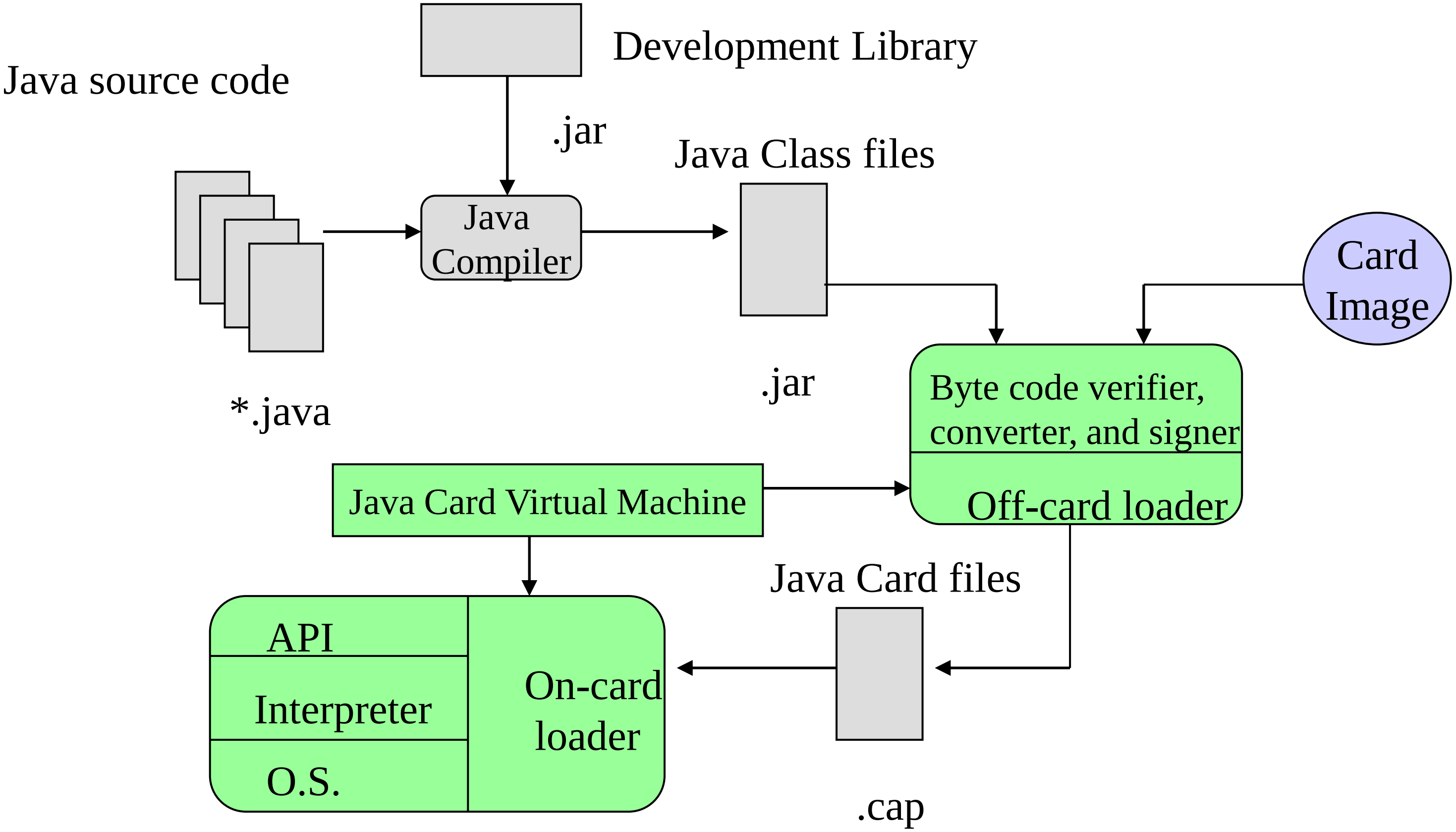
Agenda

- Introduction
- Run Time environment
- Virtual machine
- API

General Architecture

- A set of specifications
  - Issued by Sun Microsystems
  - Based on the Java platform
  - http://java.sun.com/products/javacard
- Split in three parts
  - Application programming interfaces (APIs)
  - Execution environment (JCRE)
  - Virtual machine (VM)
- Another view: architecture, memory, life cycle

Java Card Architecture



Université de Limoges

SUFOP Service Universitaire de Formation Permanente

Architecture

- Converter (part of the JDK):
  - Class file format takes too much space on java card
  - Produces a format that fits the SC 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

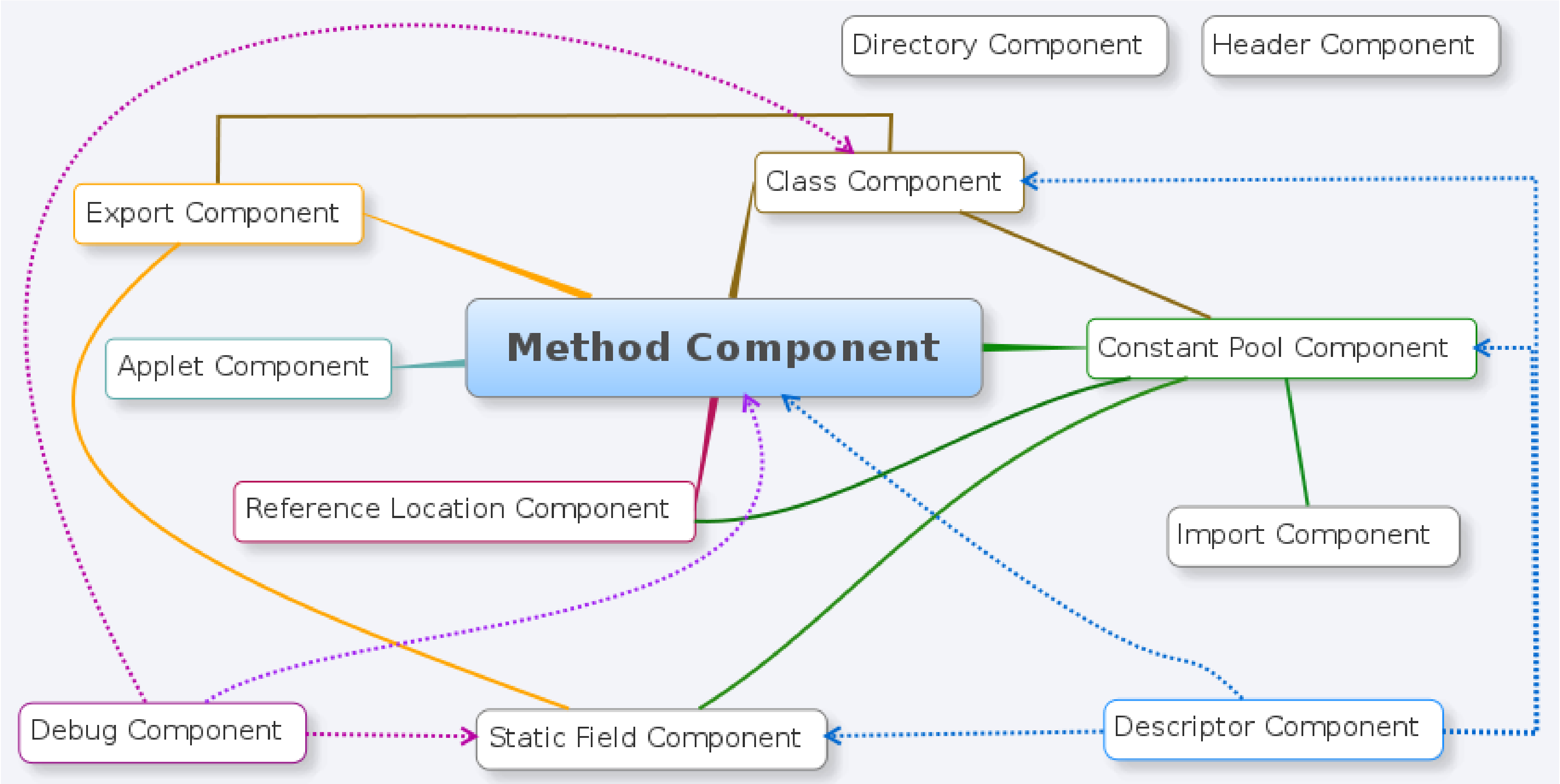
Two specific file formats

- 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 scop, class name, method signature,…)
  - Can be freely distributed, used during pre-linking phase
  - Not loaded into the card

The CAP file

- Contains an executable representation of package classes
- Contains a set of components (11)
- Each component describes an aspect of CAP file
  - Class info
  - Executable byte code
  - Linking info,…
- Optimized for small footprint by compact data structure
- Loaded on card

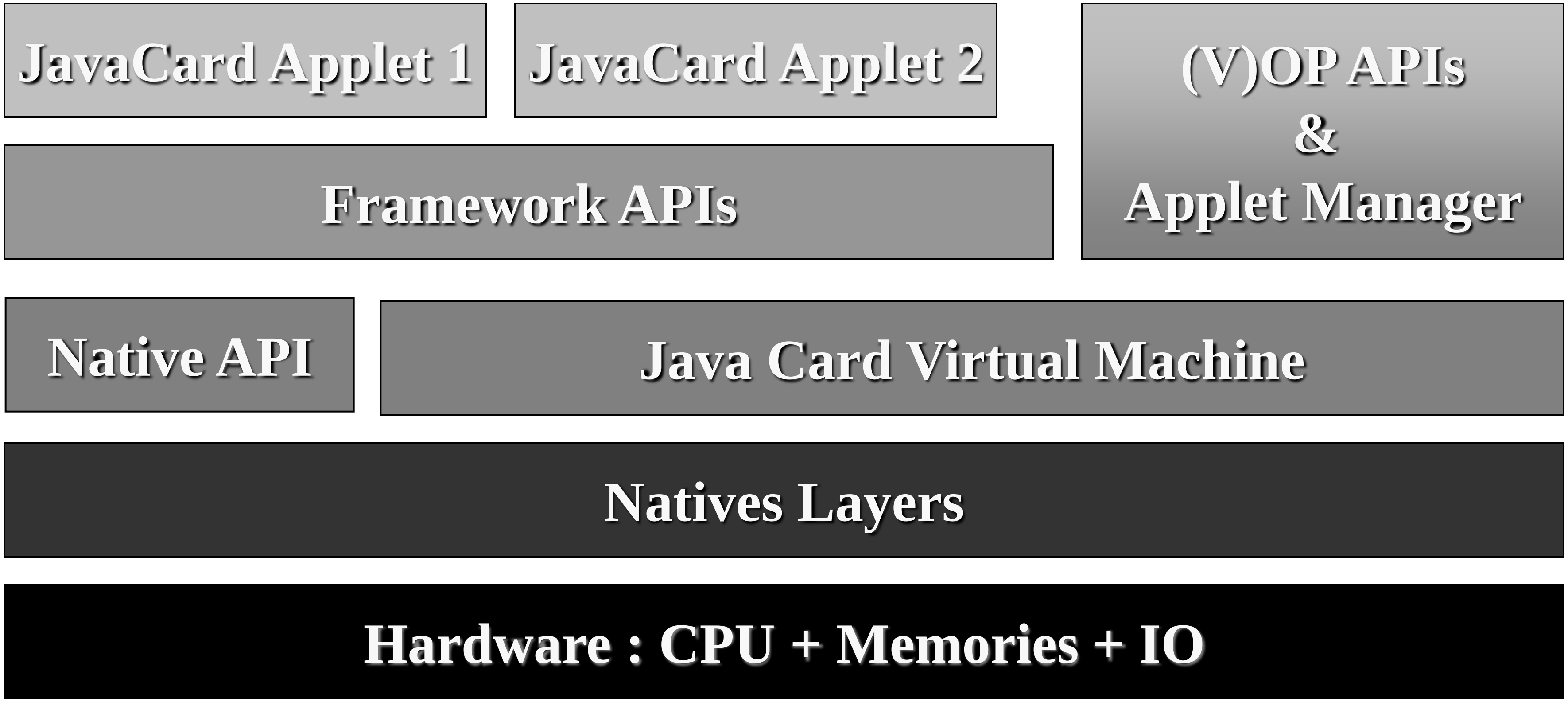
Interdependences



Université de Limoges

SUFOP Service Universitaire de Formation Permanente

Java Card architecture



JavaCard Applet 1

JavaCard Applet 2

(V)OP APIs
&
Applet Manager

Framework APIs

Native API

Java Card Virtual Machine

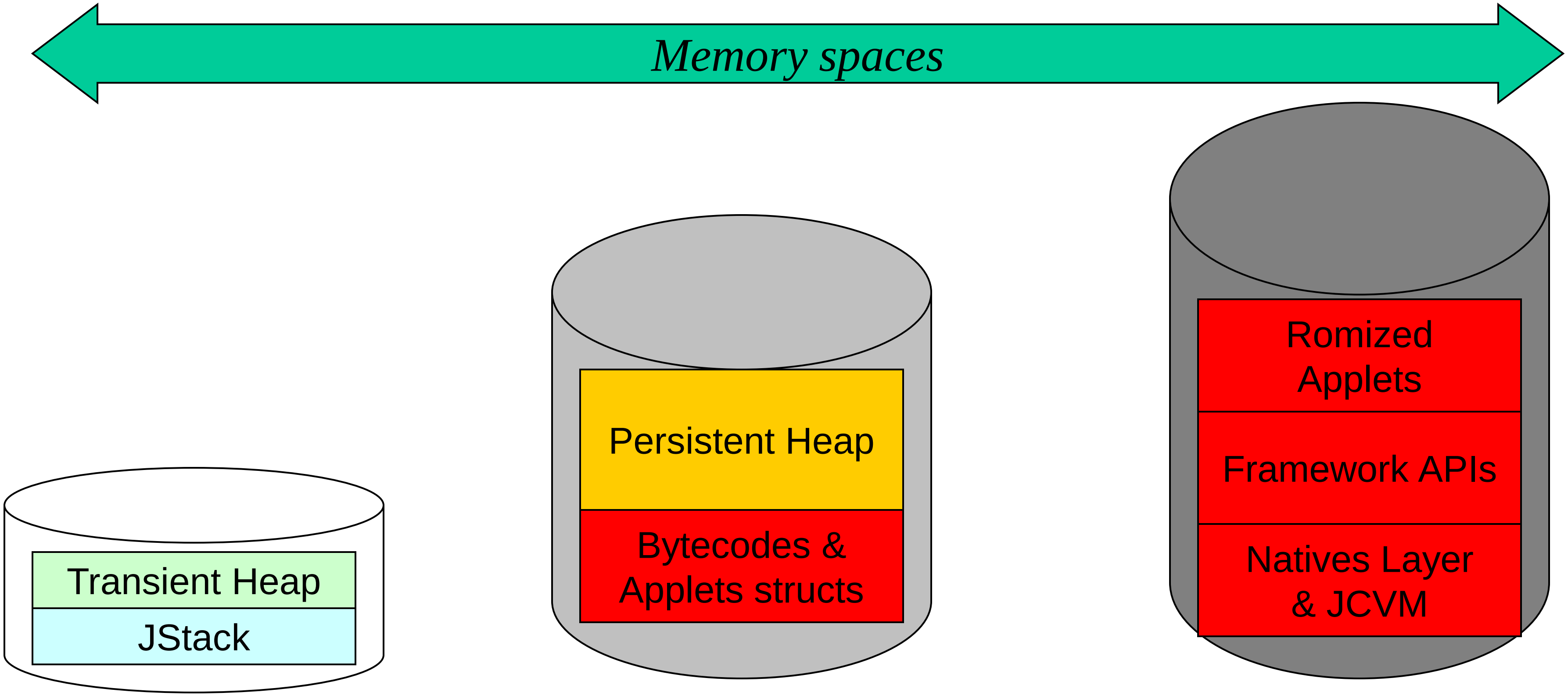
Natives Layers

Hardware : CPU + Memories + IO

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 stop 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

Java Card 3

- Roadmap
  - Published march, 31, 2008
- Integrate
  - Multithreading, Garbage Collector
  - TCP/IP, HTTP, Servlets
  - Persistence by reachability
  - Descriptive security, several VM execution mode,…
  - Sharing model of services/object ..
- Two different profiles,
  - Connected, CLDC 1.1 --/++
  - Classic i.e. JC2.2.2++
- Required architecture
  - Close to J2ME/CLDC
  - 32b, MMU/noMMU, 40KB RAM, 256KB FlashNOR

Security

- New features vs. JC
  - SecurityManager
  - AccessController
  - Java.net.SocketPermission
- New challenges
  - Update servlet and/or applet reconfiguration
  - On-line application update
  - Implementation of declarative security

Agenda

- Introduction
- Run Time environment
- Virtual machine
- API

Execution environment (JCRE)

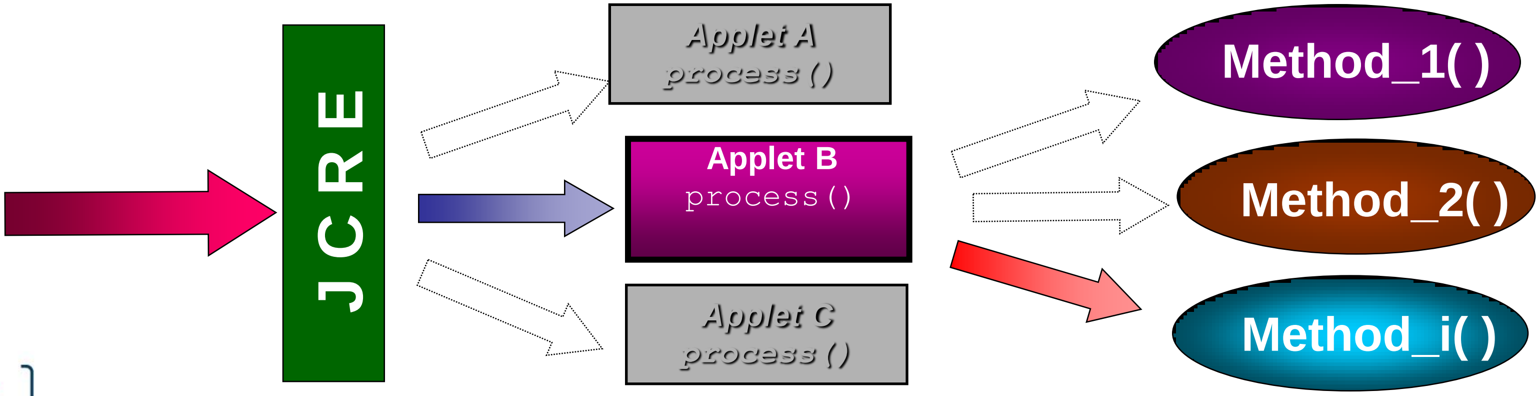
- Define how a Java Card manages its resources
- Define constraints on the Java Card operating system
  - Applet lifetime (installation, register and deletion)
  - Logical channels and applet selection,
  - Transient objects,
  - Applet isolation (firewall) and sharing,
  - Transaction and atomicity,
- 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 Gemplus
      - `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

Two Command dispatchers

- JCRE's task: main dispatcher
  - Route the incoming commands to the JCVM 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)

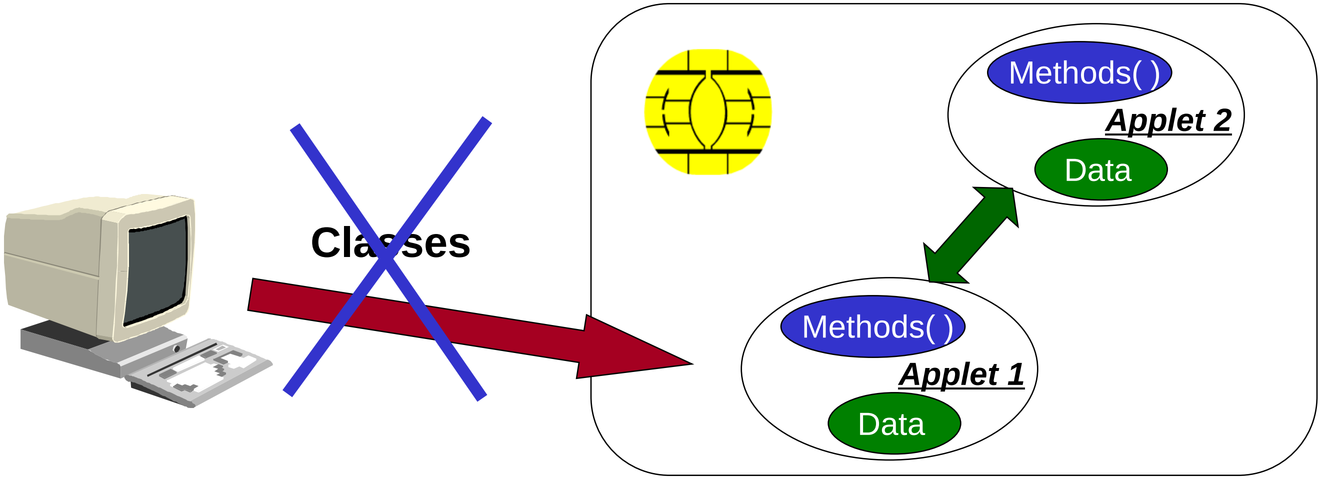


Université de Limoges

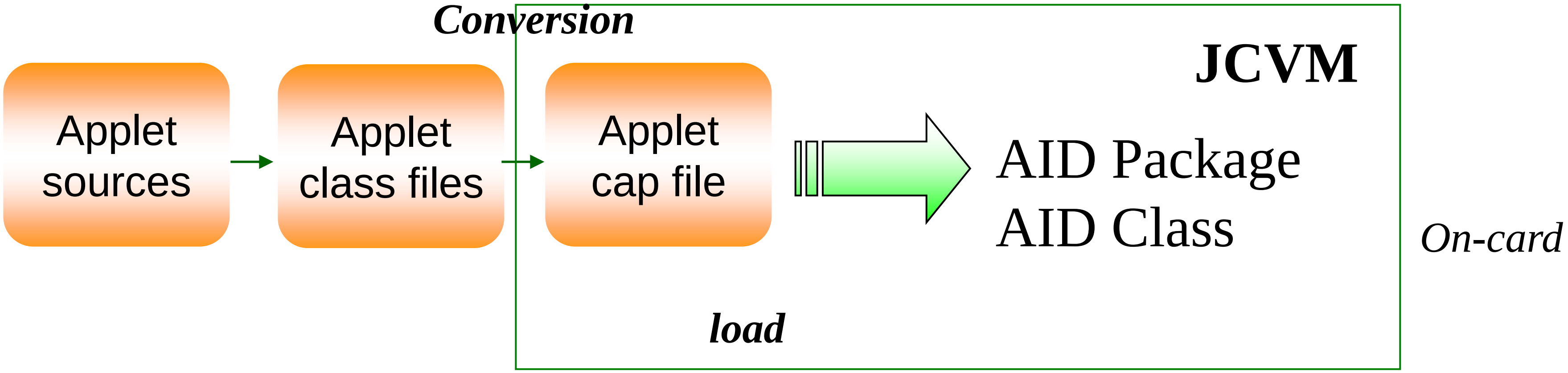
SUFOP Service Universitaire de Formation Permanente

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



Conversion

JCVM

Applet sources

Applet class files

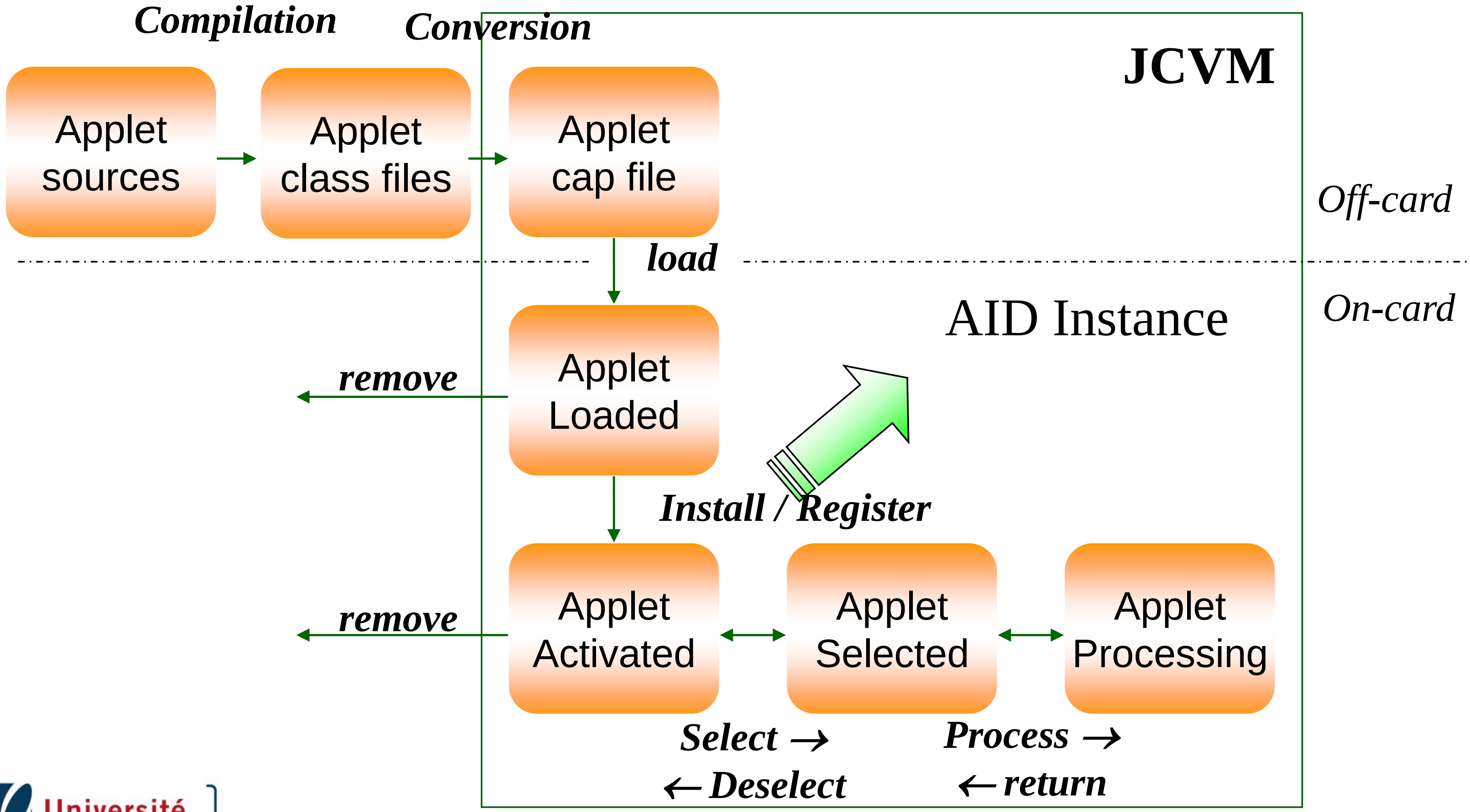
Applet cap file

AID Package
AID Class

On-card

load

Java Card applet life cycle



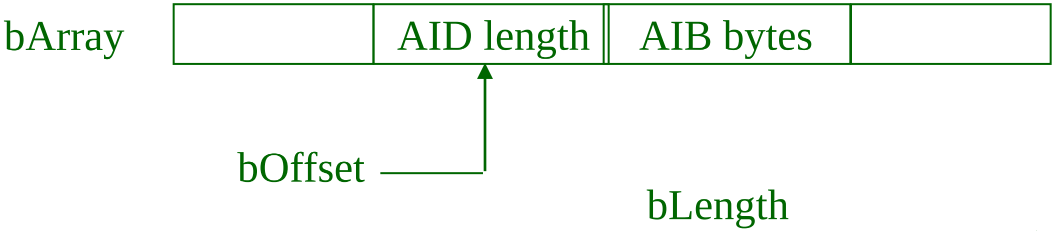
```
import javacard.framework.*
...
public class MyApplet extends Applet {
// Definitions of APDU-related instruction codes
...
// Constructor
MyApplet() {...}
// Life-cycle methods
install() {...}
select() {...}
deselect() {...}
process() {...}
// Private methods ... }
```

Install method

- JCRE call this static method first and gives
  - Applet instance AID
  - Applet privilege
  - Applet parameters
- Install method create an instance of an Applet subclass
  - Perform any necessary initializations,
  - If no parameter is provided, only one installation
- The `install`() method must directly or indirectly call the `register`() method to complete the installation; failing to do so will cause installation to fail.
- Uninstall method (JC 2.2) will be seen later.

Register method

- Used by the applet to register this applet instance
- Interacts with the java Card runtine environment
  - `void register()`
    - Assign the applet instance AID with class AID byte
  - `void register(byte [] bArray, short bOffset, byte bLenght)`
    - Assign applet instyance 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]);`



Select Method

- The JCRE invokes `select`() to notify the applet that it has been selected for APDU processing.
- You don't have to implement this method unless you want to provide session initialization or personalization.
- The `select`() method must return true to indicate that it is ready to process incoming APDUs, or false to decline selection.
- The default implementation by javacard.framework.Applet class returns true.

```
public boolean select() {
// Perform any applet-specific session initialization.
return true; }
```

Process method (1)

- Once an applet has been selected it is ready to receive command APDUs
- 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

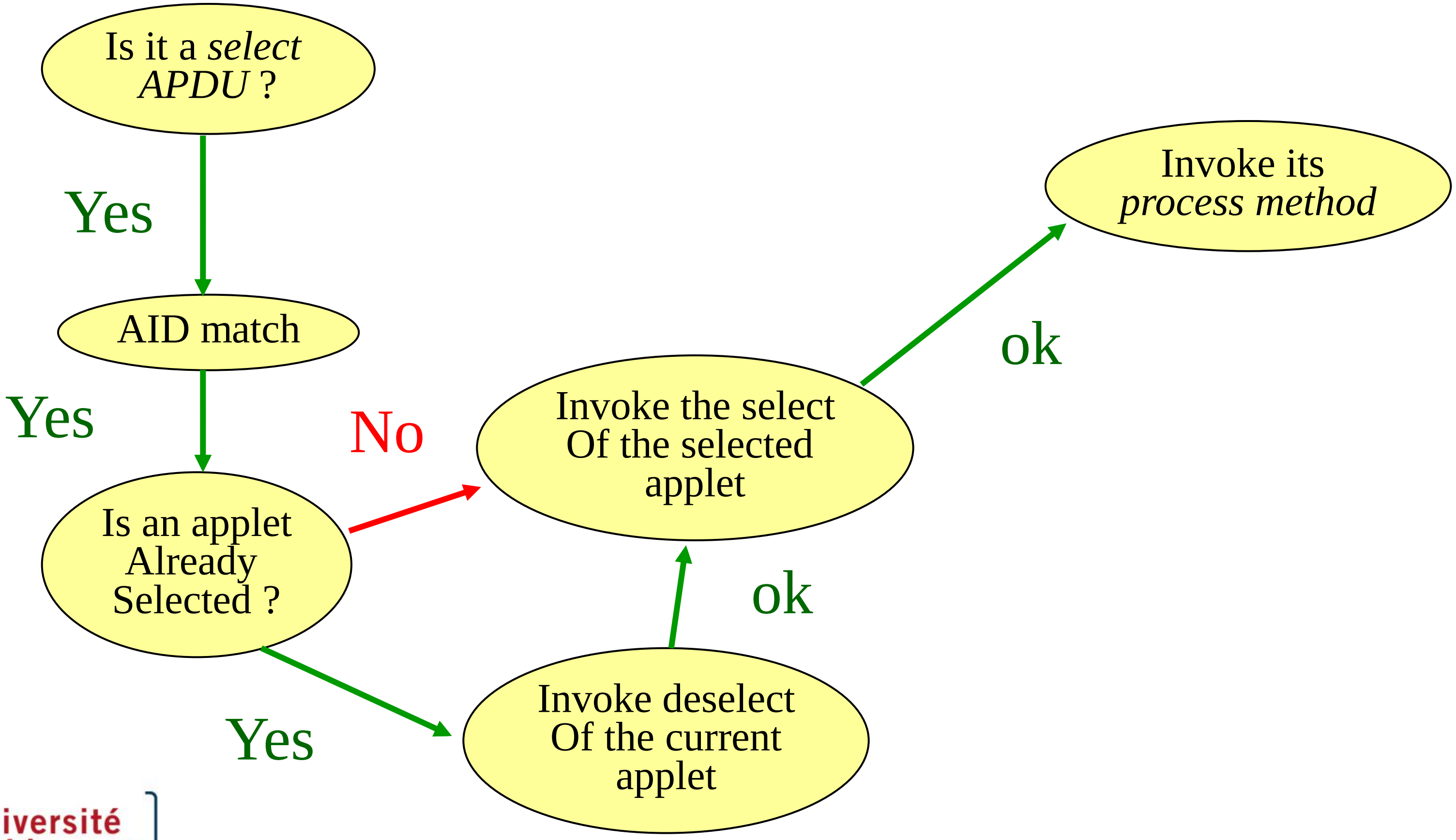
Process Method (2)

The generic process() method is:

1. Extracts the APDU CLA and INS fields
2. Retrieves the application-specific P1, P2, and data fields
3. Processes the APDU data
4. Generates and sends a response
5. Returns gracefully, or throws the appropriate ISO exception

```
public void process(APDU apdu) throws ISOException {
// Get the incoming APDU buffer.
byte[] buffer = apdu.getBuffer();
// Get the CLA; mask out the logical-channel info.
   buffer[ISO7816.OFFSET_CLA] = (byte) (buffer
   [ISO7816.OFFSET_CLA] & (byte)0xFC);
// If INS is Select, return
if ((buffer[ISO7816.OFFSET_CLA] == 0) &&
   (buffer[ISO7816.OFFSET_INS] == (byte)(0xA4)) )
return;
// If unrecognized class, return "unsupported class."
if (buffer[ISO7816.OFFSET_CLA] != MyAPPLET_CLA)
   ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
// Process (application-specific) APDU commands aimed at
   MyApplet.
switch (buffer[ISO7816.OFFSET_INS]) {
   case VERIFY_INS: verify(apdu); break;
```

Selection protocol



Is it a *select APDU* ?

Yes

AID match

Yes

Is an applet Already Selected ?

No

Invoke the select Of the selected applet

Yes

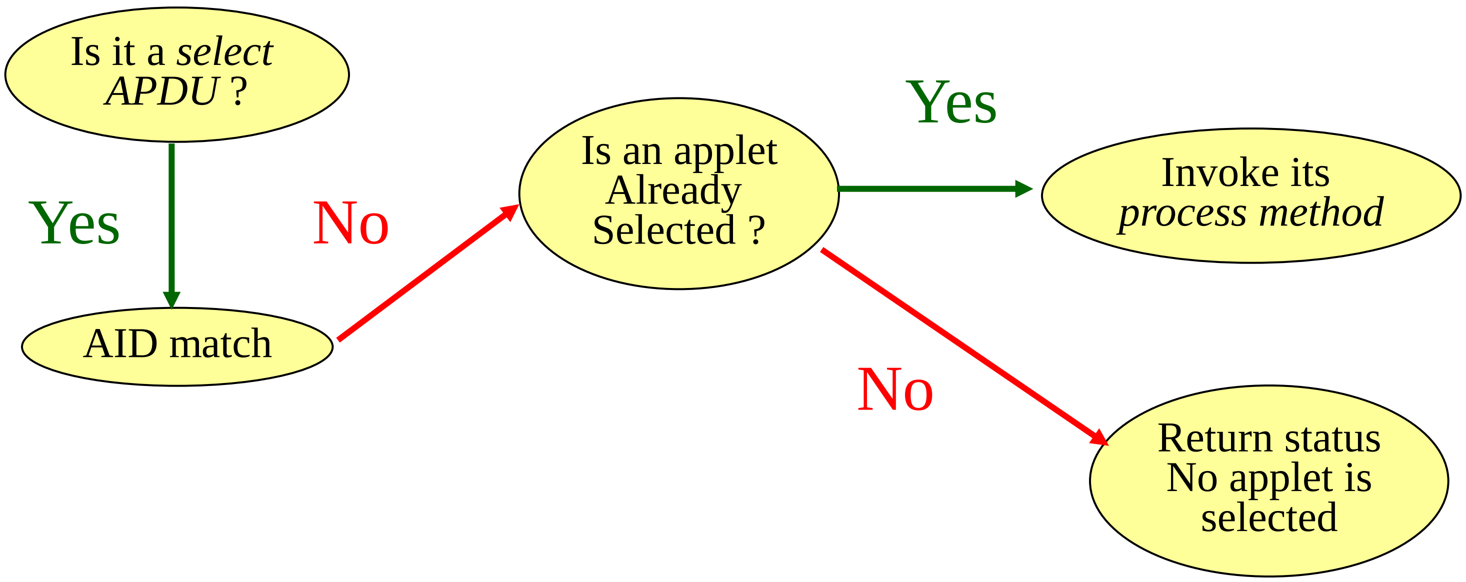
Invoke deselect Of the current applet

ok

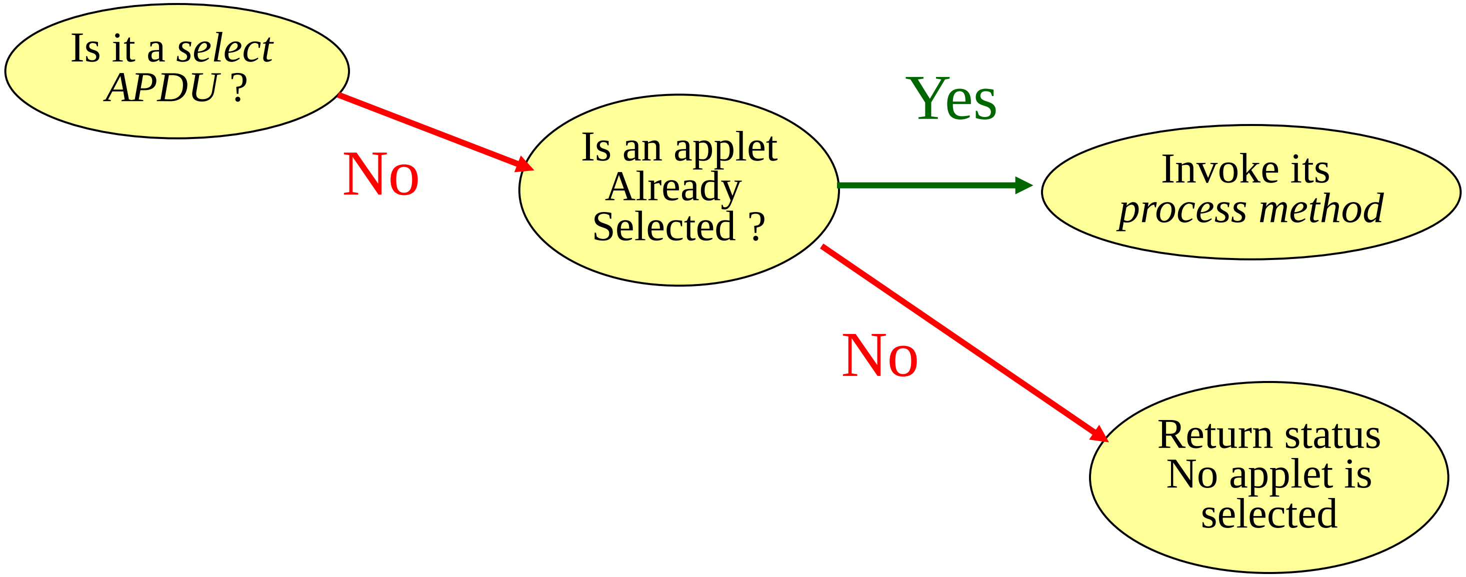
ok

Invoke its *process method*

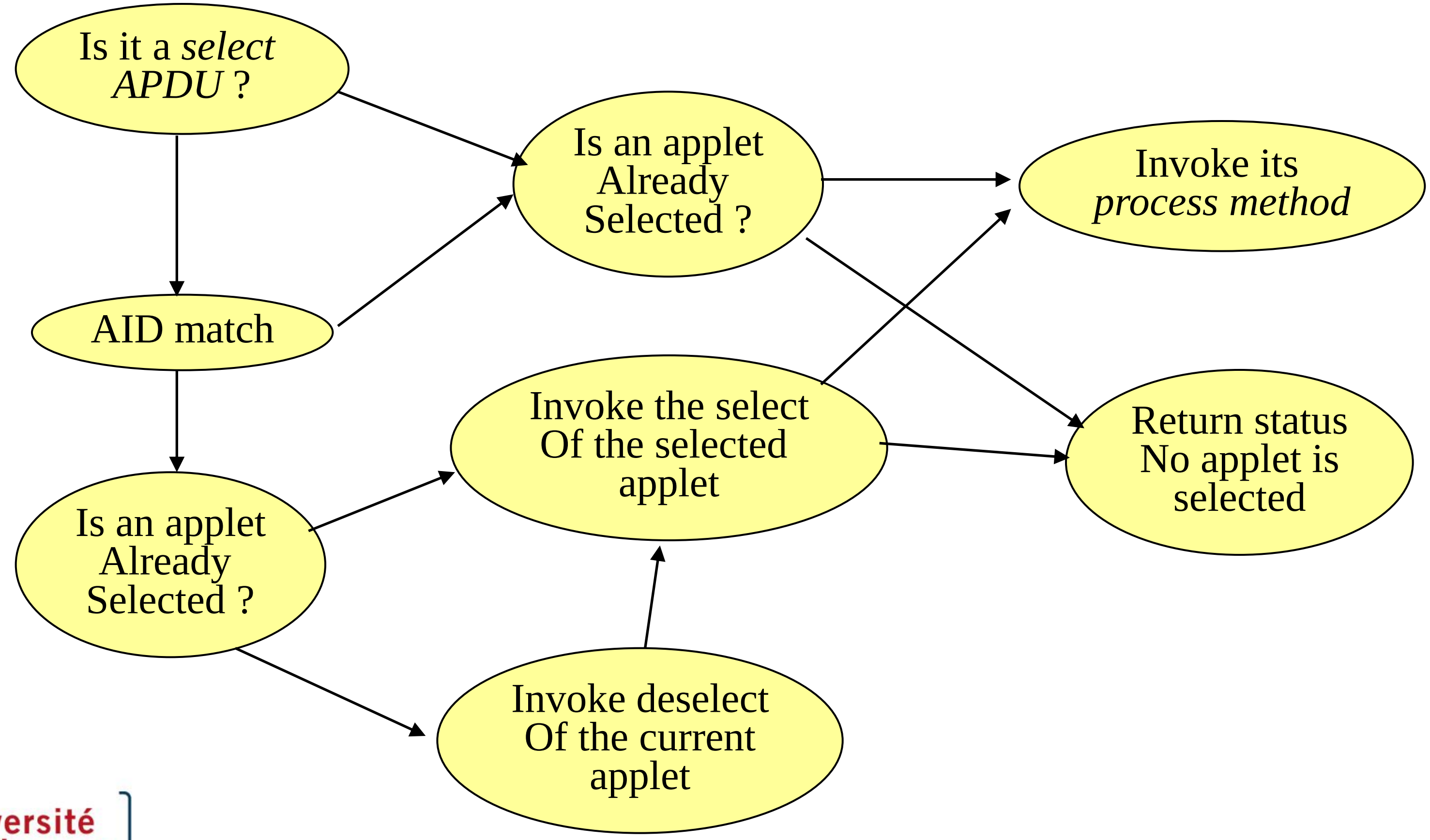
Selection protocol


Is it a *select APDU* ?
Yes
AID match
No
Is an applet Already Selected ?
Yes
Invoke its *process method*
No
Return status No applet is selected

Selection protocol


Is it a *select APDU* ?
No
Is an applet Already Selected ?
Yes
Invoke its *process method*
No
Return status No applet is selected

Selection protocol


Is it a *select APDU* ?
Is an applet Already Selected ?
Invoke its *process method*
AID match
Invoke the select Of the selected applet
Return status No applet is selected
Is an applet Already Selected ?
Invoke deselect Of the current applet

Université de Limoges

SUFOP Service Universitaire de Formation Permanente

Agenda

- Introduction
- Run Time environment
- Virtual machine
- API

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 instruction set

Java language subset

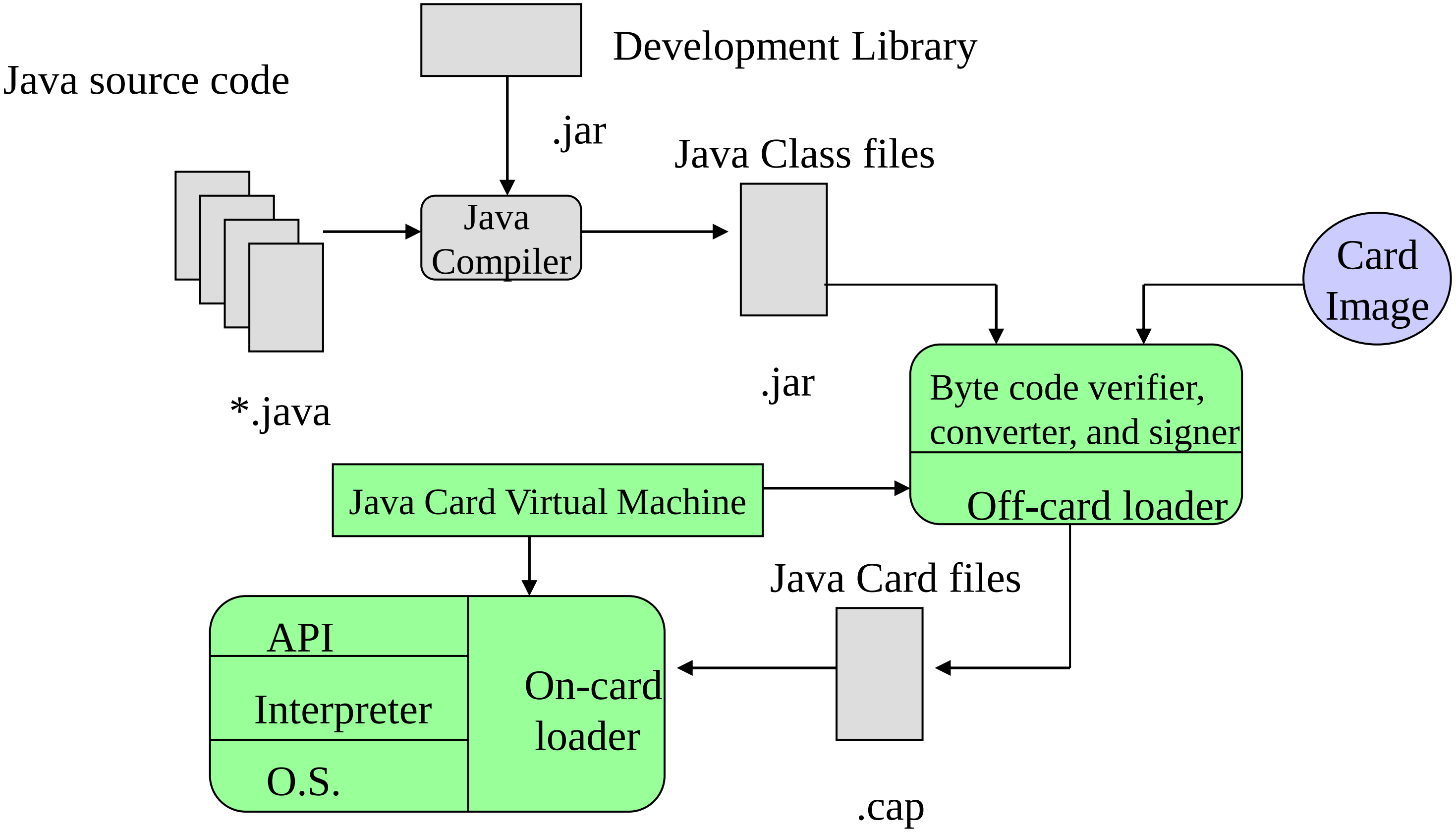
- Supported
  - boolean, byte, short
  - int (optional)
  - Objects
  - Arrays
  - Virtual methods
  - Dynamic allocation
  - Packages
  - Exceptions
  - Interfaces

- 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…

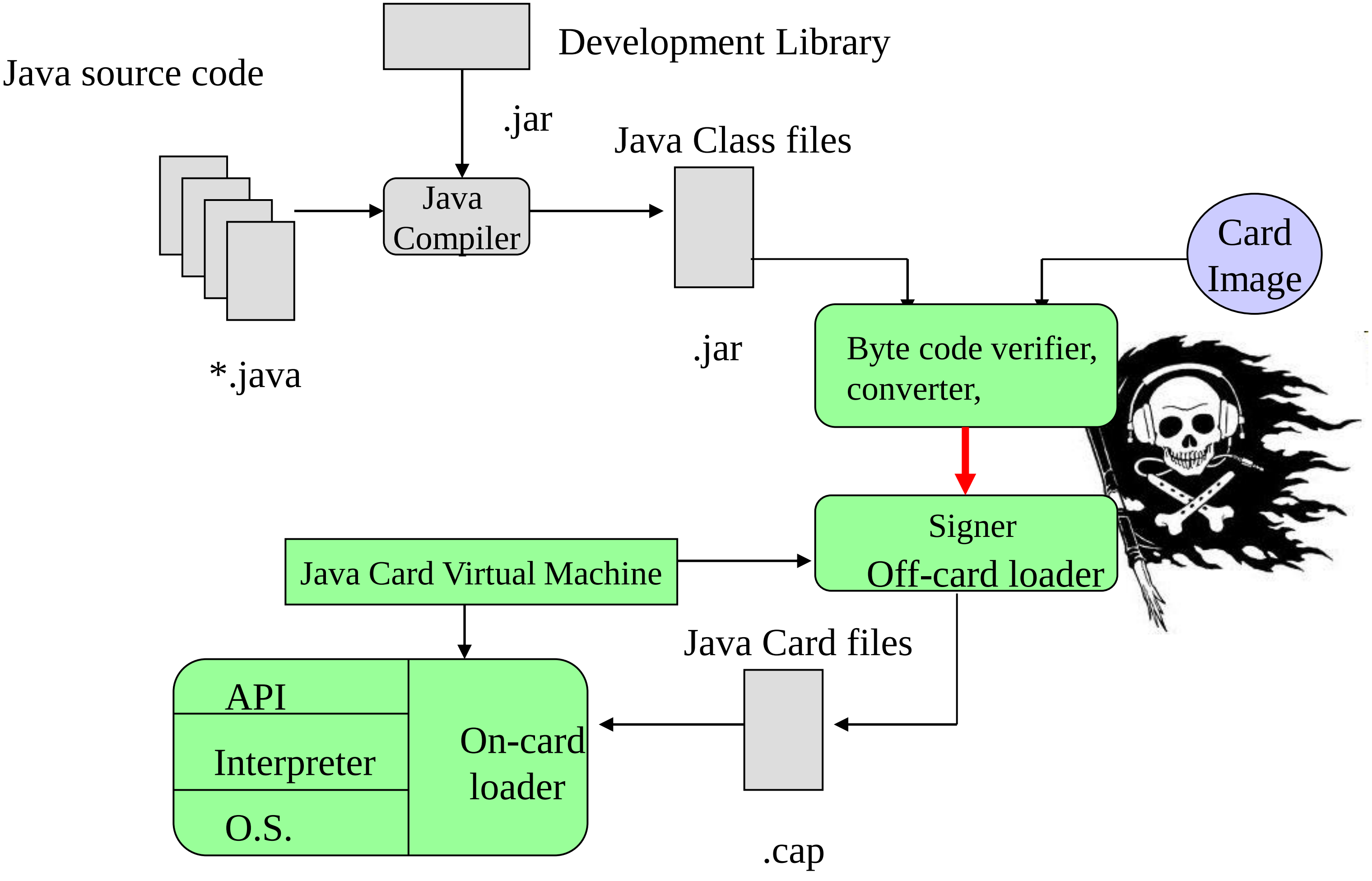
Java Card Architecture



Université de Limoges

SUFOP Service Universitaire de Formation Permanente

Java Card Architecture



Java source code

Development Library

.jar

Java Class files

Java Compiler

Card Image

*.java

.jar

Byte code verifier, converter,

Signer

Off-card loader

Java Card Virtual Machine

Java Card files

API

Interpreter

O.S.

On-card loader

.cap

Example

- Manage a counter (loyalty, ePurse) 32 bits
- APDU managed by the applet:
  - **`int read ()`**
    - command **`AA 01 XX XX 00 04`**
    - response **`RV3 RV2 RV1 RV0 90 00`**
  - **`int increment ()`**
    - command **`AA 02 XX XX 04 AM3 AM2 AM1 AM 04`**
    - response **`RV3 RV2 RV1 RV0 90 00`**
  - **`int decrement ()`**
    - command **`AA 03 XX XX 04 AM3 AM2 AM1 AM 04`**
    - response **`RV3 RV2 RV1 RV0 90 00`**

Applet counter

```
package unilim.counter ;
import javacard.framework.* ;

public class counter extends Applet {
  // valeur is 32 bit value
  private int valeur;
  public Counter() {
    valeur = 0;
    // only one instance of counter
    register(); }
  public static void install( APDU apdu ){
    new Counter(); }
}
```

Applet counter

```
public void process( APDU apdu ) {
 /* get the APDU buffer to obtain a reference to the APDU buffer. When the applet receives
        the APDU  object,  only the first five APDU header bytes are available in the   APDU
buffer. They are the  CLA, INS, P1, P2, and P3 bytes respectively.
     byte[] buffer = apdu.getBuffer();

 // ignore the applet select command dispached to the process
     if (selectingApplet()) { return; }
/*When an error occurs, the applet may decide to terminate the process, and to throw an
        exception containing the status word (SW1, SW2) to indicate the     processing state  of
the card. An exception that is not caught by an applet is  caught by the JCRE.    */
     if ( buffer[ISO7816.OFFSET_CLA] != 0xAA )
       ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
     switch ( buffer[ISO7816.OFFSET_INS] ) {
     case 0x01: ... // read
     case 0x02: ... // increment
     case 0x03: ... // decrement see next slide
     default:
       ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED); }
   }
}
```

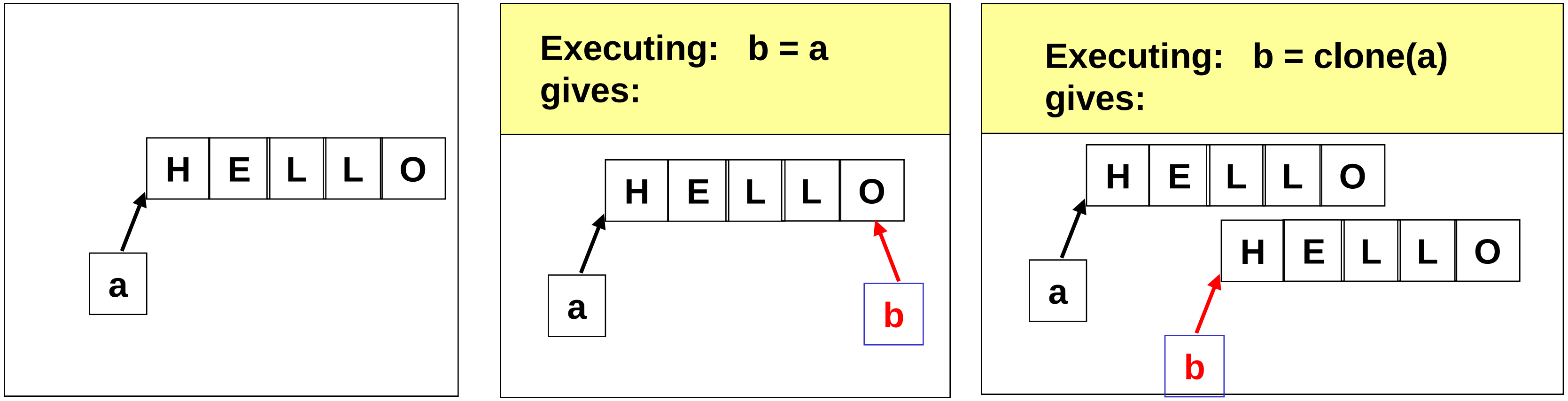
Decrement

```
case 0x03: // Decrement
{ // receiving the data
  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];
  // treatment
  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);
  // single operation
  apdu.setOutgoingAndSend((short)0, (short)4);
  return;
}
```

Agenda

- Introduction
- Run Time environment
- Virtual machine
- API

Unsupported features : Cloning



- No `clone()` method in class Object
  Use `Util.arrayCopy()` instead
- Clonable interface not provided
- Creating copies of objects is critical

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 does 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, ISO, PIN, 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

- **`public final class AID`**

  Encapsulates constants related to ISO7816-5

  Public final static field with SW_ prefixes in order to standardize status response

  Fields with OFFSET_ prefixes defines constants to use as index in the APDU buffer byte array

- **`public class ISOException`**

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

  throwIt () allows to output a status word error

- **`public class Util`**
  - Byte array manipulation and comparaison
  - Type conversion (short/byte)
    - **`arrayCopy`** () atomic/non-atomic copy of byte array
    - **`arrayCopyNonAtomic`** () non-atomic copy of byte array
    - **`arrayCompare`** () byte array comparaison
    - **`arrayFillNonAtomic`**() : fill an array with an int value
    - **`makeShort`** () create a short using a byte

Javacard.framework

public class ISO7816

- ISOExceptioon.throwtIt (short reason)
- public abstract class PIN
  - Represent an byte array for a PIN code
  - ownerPIN secret code to be update and/or checked
- Static constant field
  - **public static final short SW_NO_ERROR = (short) 0x9000**
  - **public static final short SW_FILE_NOT_FOUND = (short) 0x6A82**
  - **public static final short SW_RECORD_NOT_FOUND = (short) 0x6A83**
  - **public static final short SW_INCORRECT_P1P2 = (short) 0x6A86**
  - **public static final short SW_WRONG_P1P2 = (short) 0x6B00**
  - **public static final short SW_CLA_NOT_SUPPORTED = (short) 0x6E00**
  - **...**
  - **public static final byte CLA_ISO7816= 0x00**
  - **public static final byte INS_SELECT= 0xA4**
  - **public static final byte INS_EXTERNAL_AUTHENTICATE= 0x82**

Pin interface

- Define a PIN behavior and a PIN value
- Try limit maximum trial 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.

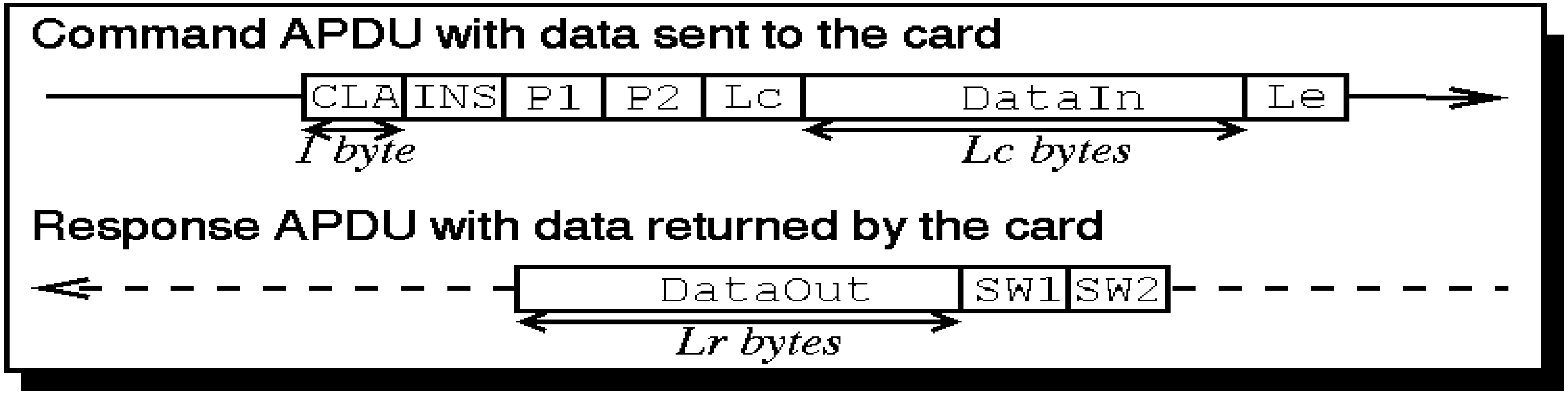
Javacard.framework.JCSystem

`public final class JCSystem`

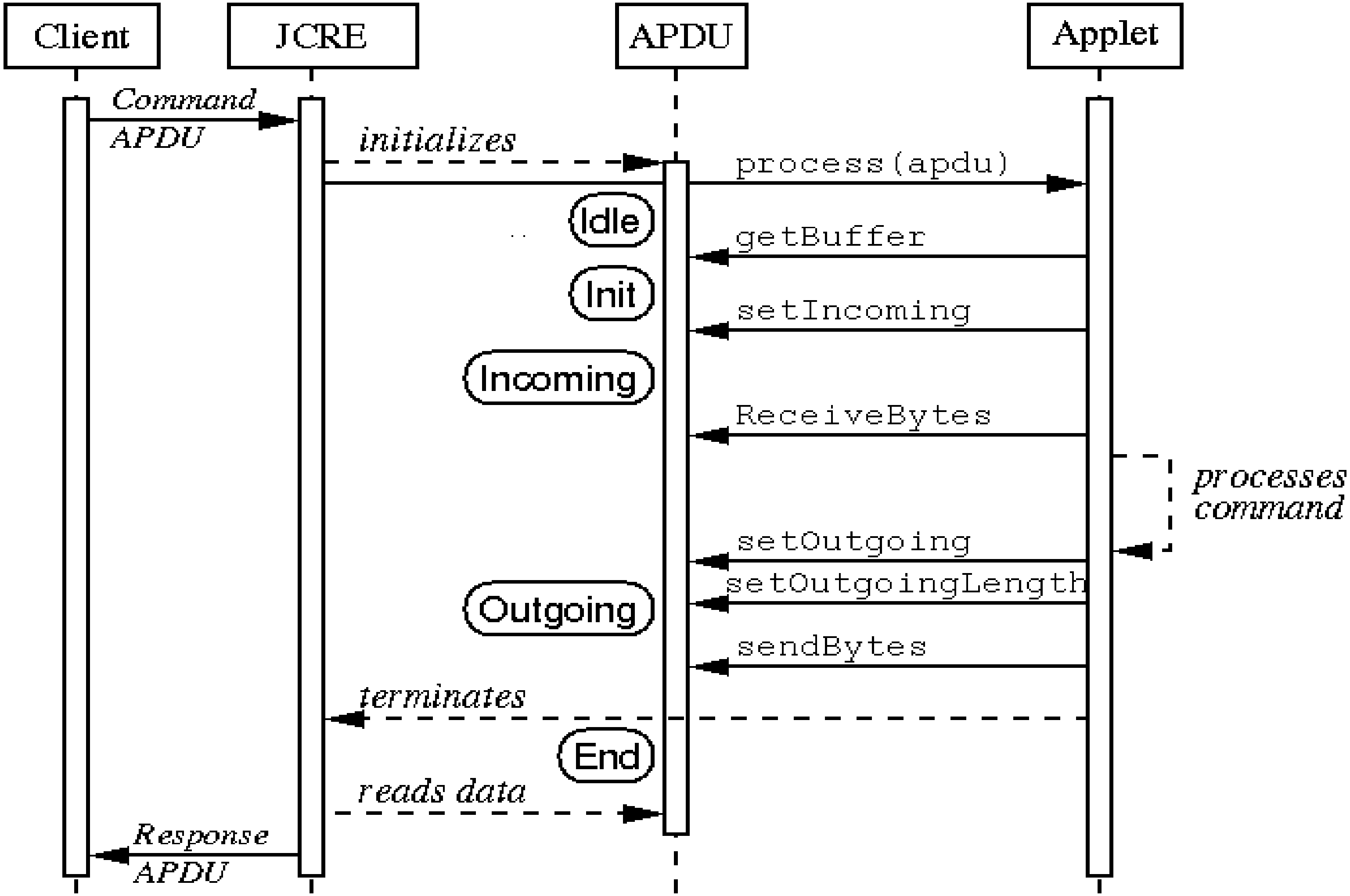
- Static methods (natives) for interaction with the JCRE
- Transactions
  - Only one level of transaction allowed
  - **`beginTransaction(), commitTransaction (), abortTransaction ()`**
  - Limited by the ram and eeprom capacity
- Transient array
  - **`makeTransientXXXArray(lenght, event)`**
  - Build a transient array reinitialized at reset or de-selection
- Object sharing
  - Build an object that inherit from **`shareable`**
  - Provide a reference with **`getAppletShareableInterfaceObject (AID, Parameter)`**

Javacard.framework.APDU

- Define the format of the «data packets» exchanged between a reader and a card:
  - APDUs (Application Programming Data Units) are defined for both commands and responses
  - 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 that the APDU buffer size and need to be read/write in portions by the applet.


Command APDU with data sent to the card
CLA
INS
P1
P2
Lc
DataIn
Le
1 byte
Lc bytes
Response APDU with data returned by the card
DataOut
SW1
SW2
Lr bytes

Handling the **APDU** class



Univ de Li

SUFOP Service Universitaire de Formation Permanente

Unfolding of a communication Inbound

- The JCRE receives the APDU
- The JCRE fills the `APDU` buffer with the header of the APDU
- The JCRE calls the `process()` method of the selected applet
  - with the `APDU` object in parameter
- The applet retreives the `APDU` buffer (`getBuffer()` method)
  - reads/analyses the header
  - eventually calls the `APDU` object' methods to receive data

APDU class: receiving data

- `Byte[] getBuffer()`
  - returns the APDU buffer byte array, filled withAPDU header
- `short setIncomingAndReceive()`
  `throws APDUException`
  - sets the transfer direction to inbound
  - receives the incoming data in the APDU buffer at offset = 5
- `short receiveBytes(short bOff)`
  `throws APDUException`
  - sets the transfer direction to inbound
  - receives the incoming data in the APDU buffer at user offset

Université de Limoges

SUFOP Service Universitaire de Formation Permanente

Unfolding of a communication Outbound

- The applet processes the APDU
  - SW1||SW2 = 90 00 by default
  - At any point, the applet can send an ISOException by invoking the static method `ISOException.throwIt(reason)`
    - The JCRE automatically sends back the corresponding status word
- Eventually, the applet constructs an APDU response
  - by filling the `APDU` buffer
  - by invoking methods on the `APDU` object
- The JCRE sends back the response

APDU class: sending back data

- `Short setOutgoing()throws APDUException`
  - Sets the transfer direction to outbound
  - Obtains the length Le expected by the terminal
- `setOutgoingLength(short length) throws APDUException`
  - Sets the actual length
- `sendBytes(short offset, short length)throws APDUException`
  - Sends `length` bytes from the APDU buffer starting at `offset`

APDU class: sending back data (2/2)

- `setOutgoingAndSend(short offset, short length) throws APDUException`
  - `setOutgoing() + setOutgoingLength() + sendBytes`

- `sendBytesLong(byte[] array, short offset, short length) throws APDUException`
  - Sends `length` bytes from another byte *array* starting at `offset`

Javacard.framework.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
  - `void process(APDU apdu)`
    - Called for requesting the applet isntance to execute a receive command
- User applets may overrides
  - `protected void final register()`
    - Called during installation for registering the newly created instance
  - `boolean select()`
    - Called when the applet instance is selected
  - `void deselect()`
    - Called when the applet instance is selected

`javacard.security` package

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

2.2.2 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

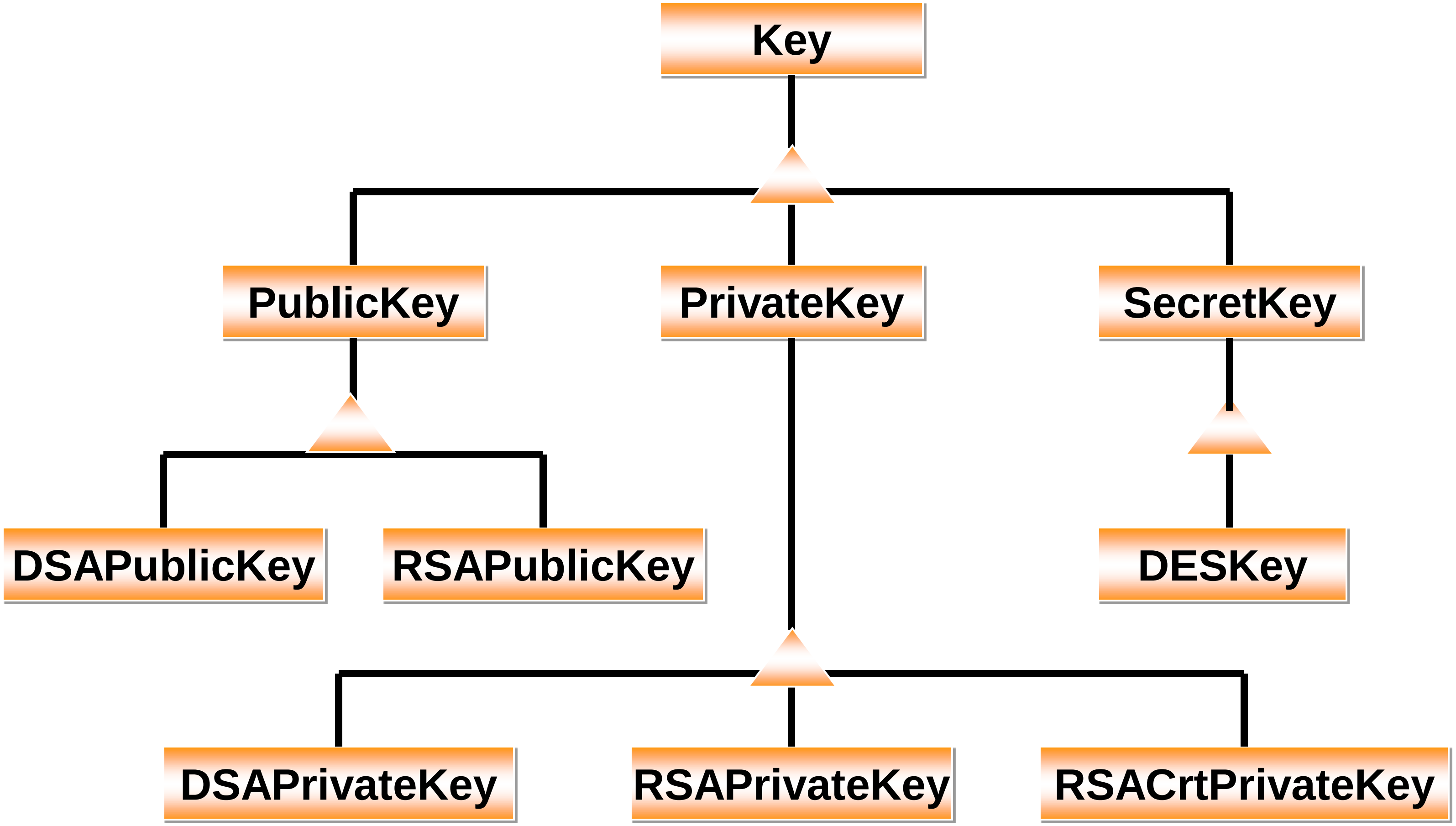
AES: Advanced Encryption Standard (AES), as defined by NIST in FIPS 197

HMAC: Keyed-Hashing for Message Authentication, as defined in RFC-2104

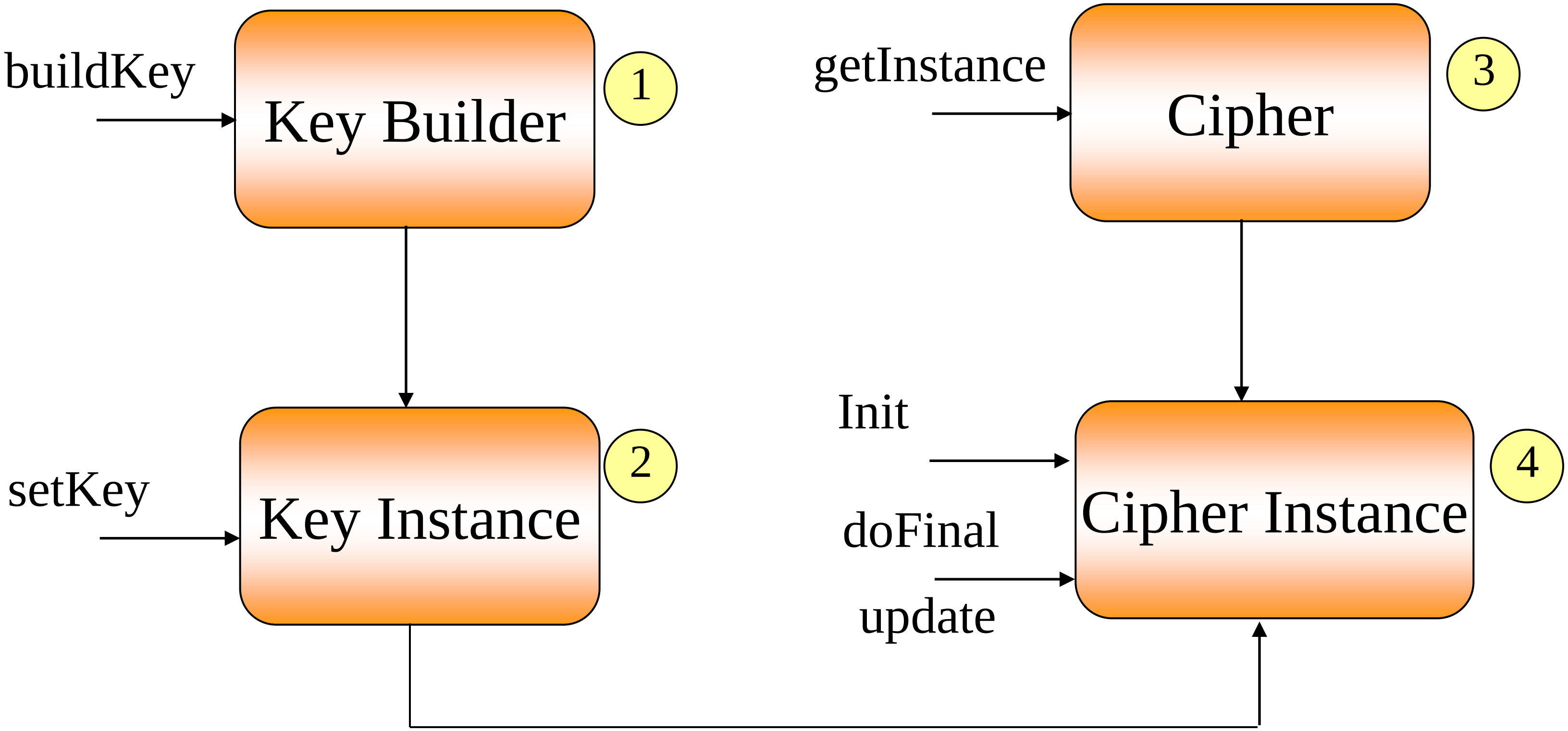
And some more…

- 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.

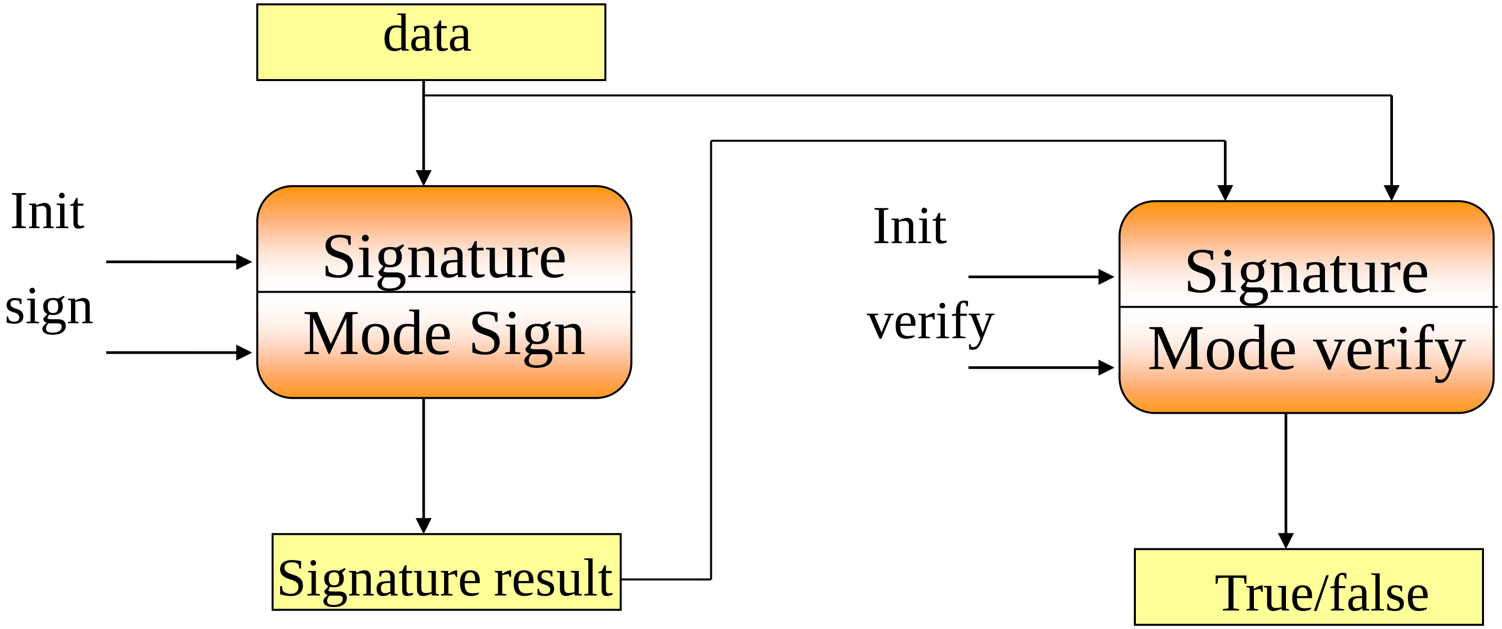
The key hierarchy



Principe


buildKey
Key Builder
1
getInstance
Cipher
3
setKey
Key Instance
2
Init
doFinal
update
Cipher Instance
4

Principe


data
Init
sign
Signature
Mode Sign
Signature result
Init
verify
Signature
Mode verify
True/false

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

Cryptography usage

- Once inside the applet:

```
DESKey k = KeyBuilder.buildKey( TYPE_DES, LENGTH_DES,
  false) ;
Signature s = Signature.getInstance( ALG_DES_MAC_NOPAD,
  false);
k.setKey(buffer, offset, length) ;
```

- At each usage:

```
s.init(k, MODE_SIGN) ;
s.update(in_buff, in_ofs, in_len) ;
s.sign(in_buff, in_ofs, in_len, out_buff, out_ofs) ;
s.verify(in_buff, in_ofs, in_len,out_buff, out_ofs);
```

javacardx.security.cipher

- Defines all cipher algorithms methods management
- Implementation of Cipher algorithms must extend this class and implement all the abstract methods
- Contains all cipher constants value for getInstance algorithm parameter specification
- Once inside the applet:

```
DESKey k = KeyBuilder.buildKey( TYPE_DES, LENGTH_DES,
  false) ;
c = cipher.getInstance( ALG_DES_MAC_NOPAD, false);
k.setKey(buffer, offset, length) ;
```

- At each usage:

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

Any question ?

Université
de Limoges

SUFOP Service Universitaire
de Formation Permanente