



16/12/2024

***Rapport de Projet :  
Carte à Puce Java pour  
Machine ATM***

# SOMMAIRE

TP1 -Environnement de développement JavaCard (v 2.0).....	3
But de ce TP :.....	3
I.    Installation du JDK .....	3
II.    Installation de l'IDE - Eclipse :.....	5
III.    L'indispensable « Hello Word » .....	5
IV.    Installation du Java Card Development Kit 2 .2.2 : .....	8
TP2 – Développement d'une application coté Serveur (V2.0).....	10
But de ce TP.....	10
1.    Création de l'applet card sous Eclipse :.....	10
2.    Codage de notre applet : .....	13
3.    Outils de simulation : .....	15
TP3 - Programmation d'une application coté client.....	20
But de ce TP .....	20
I.    Création de l'application client sous Eclipse .....	20
1.    Création d'un nouveau projet .....	20
2.    Ajout de la librairie « apduio » dans le classpath .....	21
3.    Création de la classe principale .....	22
•    Étape 1 - Connexion : .....	25
•    Étape 2 – Sélection : .....	25
•    Étape 3 - Invocation des services implémentés : .....	25
•    Étape 4 - Mise hors tension :.....	26
II.    Utilisation de l'application cliente avec un simulateur – JCWDE.....	26
Mini-Projet.....	31
I.    Partie Serveur.....	31
1.    Introduction générale.....	31

2.	Étape 1. Déclarer les attributs et les constantes .....	31
3.	Étape 2. Définition des méthodes publiques qu'elle doit obligatoirement implémenter .....	32
II.	Partie Client.....	34
1.	Introduction générale.....	34
2.	Connexion .....	34
3.	Sélection .....	35
4.	Invocation des services implémentes .....	36
5.	Mise hors tension .....	40
6.	Create apdu /sendapdu .....	40
7.	Verif PIN .....	41
8.	Cryptage .....	44
9.	Préparation de l'interface graphique : .....	45

# TP1 -Environnement de développement JavaCard (v 2.0)

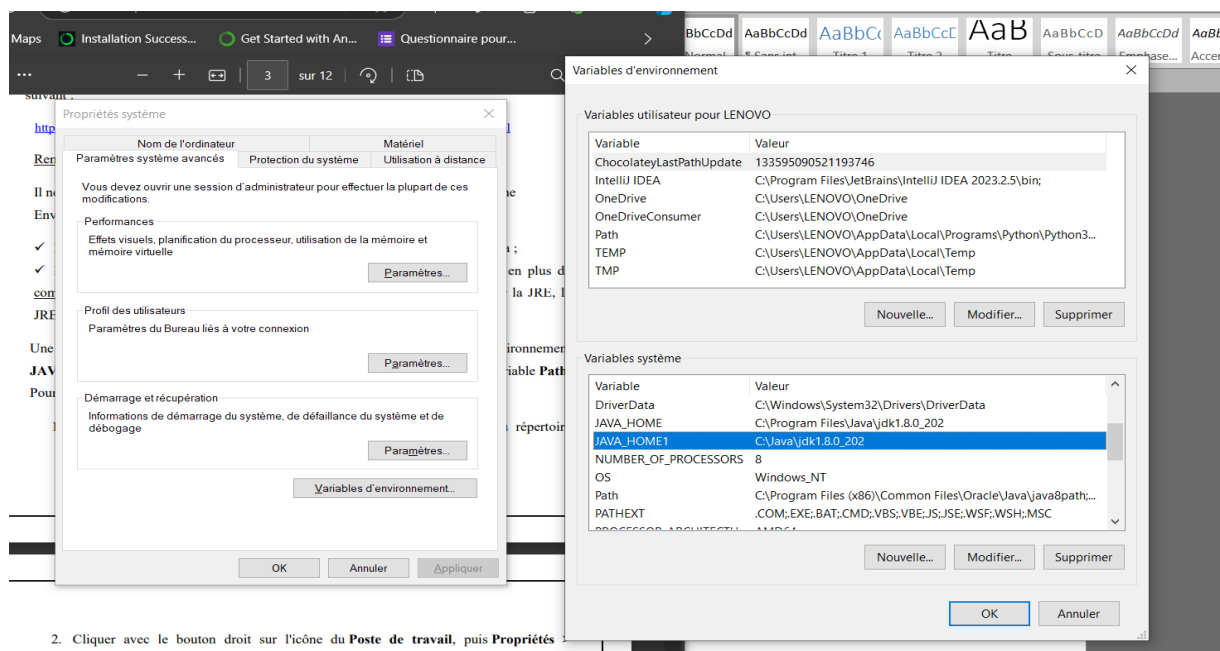
## But de ce TP :

Ce TP a pour but d'installer l'environnement de développement nécessaire pour programmer des applications JavaCard. Nous allons installer Eclipse l'IDE (Environnement de développement intégré), la plate-forme JavaCard 2.2.2 (kit de développement) ainsi que le plug-in Eclipse-JCDE (interface entre la plate-forme JavaCard et Eclipse).

## I. Installation du JDK

```
C:\Users\LENOVO>java -version
java version "1.8.0_421"
Java(TM) SE Runtime Environment (build 1.8.0_421-b09)
Java HotSpot(TM) Client VM (build 25.421-b09, mixed mode, sharing)
```

- ajouter la variable d'environnement JAVA\_HOME

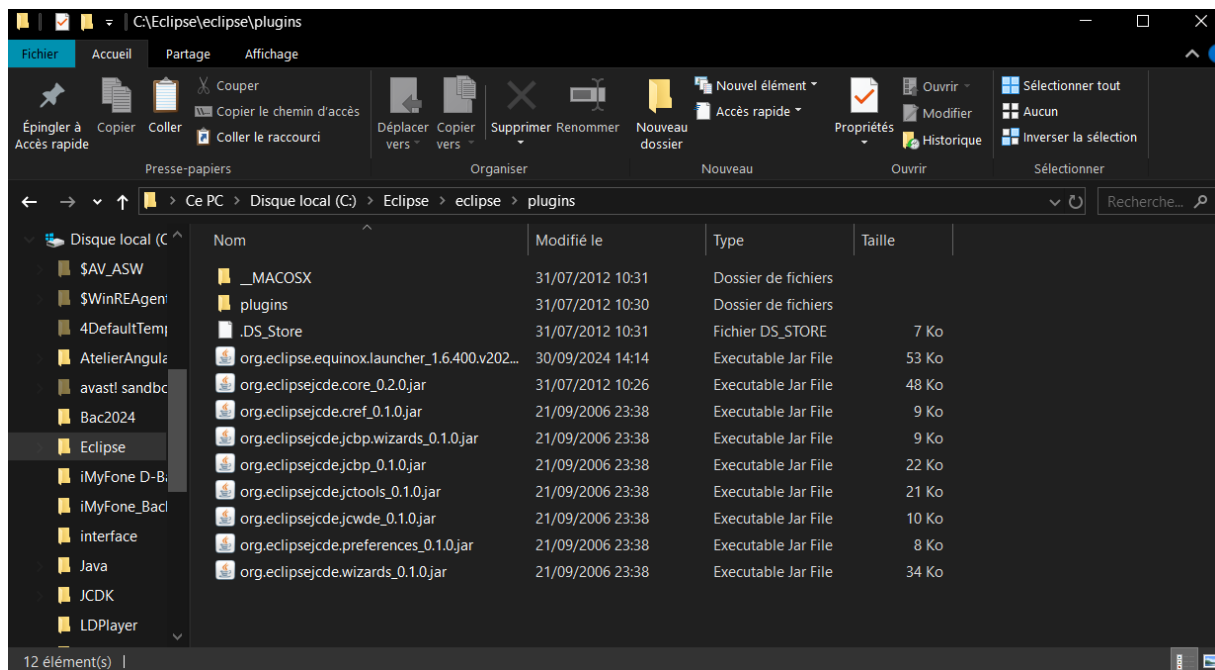


- vérifier le bon fonctionnement de votre JDK.

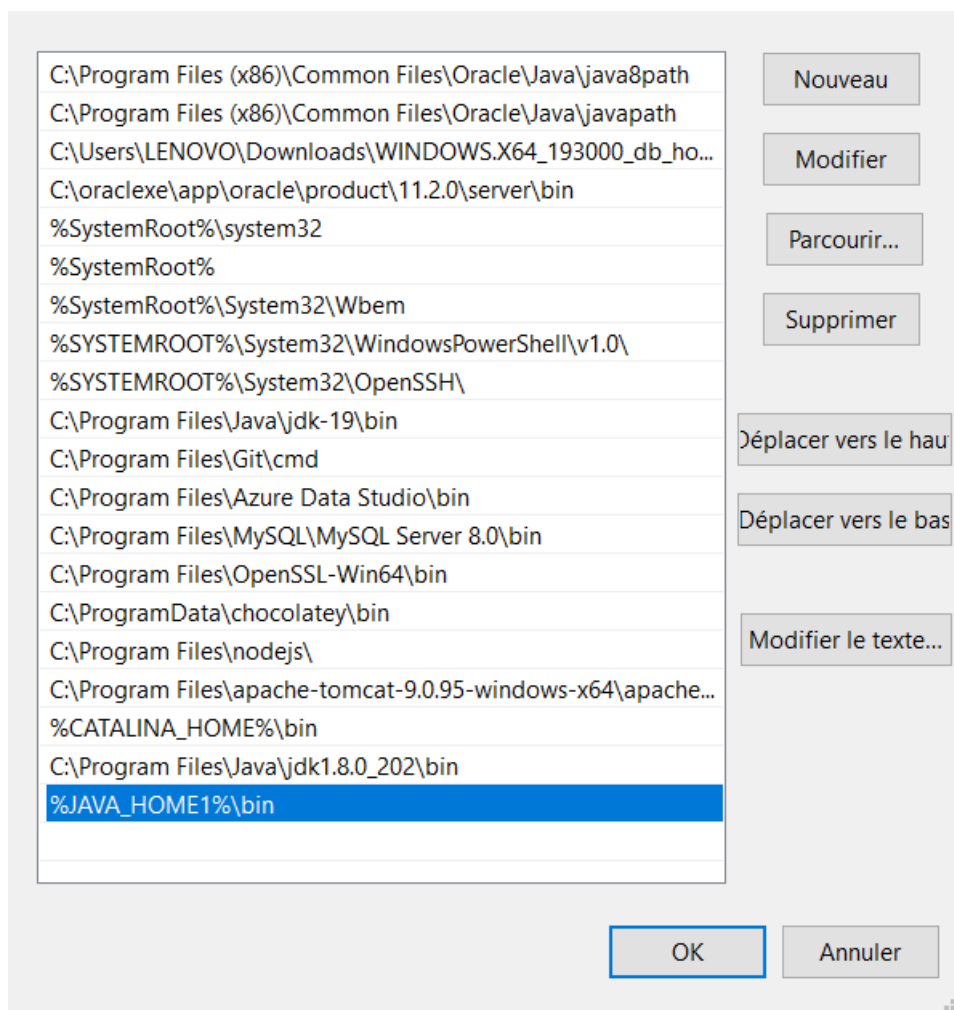
```
C:\Windows\System32\cmd.exe
Microsoft Windows [version 10.0.19045.4894]
(c) Microsoft Corporation. Tous droits réservés.

C:\JCDK\java_card_kit-2_2_2\bin>apdutool
Java Card 2.2.2 APDU Tool, Version 1.3
Copyright 2005 Sun Microsystems, Inc. All rights reserved. Use is subject to license terms.
Opening connection to localhost on port 9025.
java.net.ConnectException: Connection refused: connect

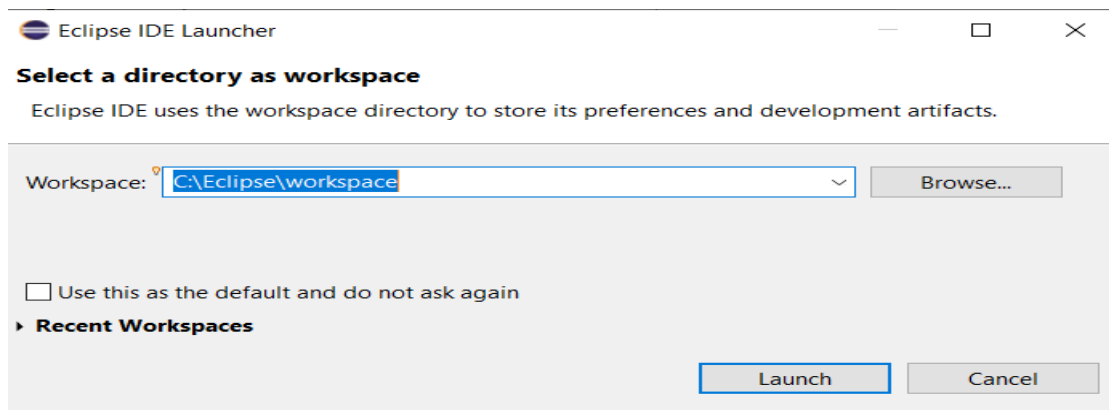
C:\JCDK\java_card_kit-2_2_2\bin>
```



### Modifier la variable d'environnement

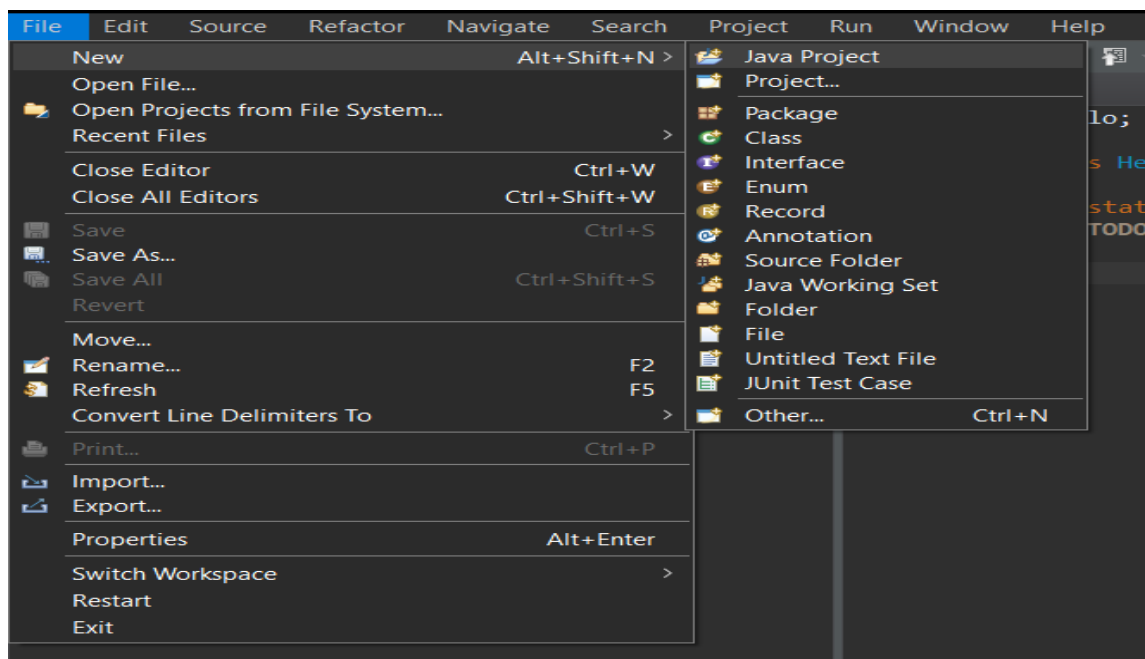


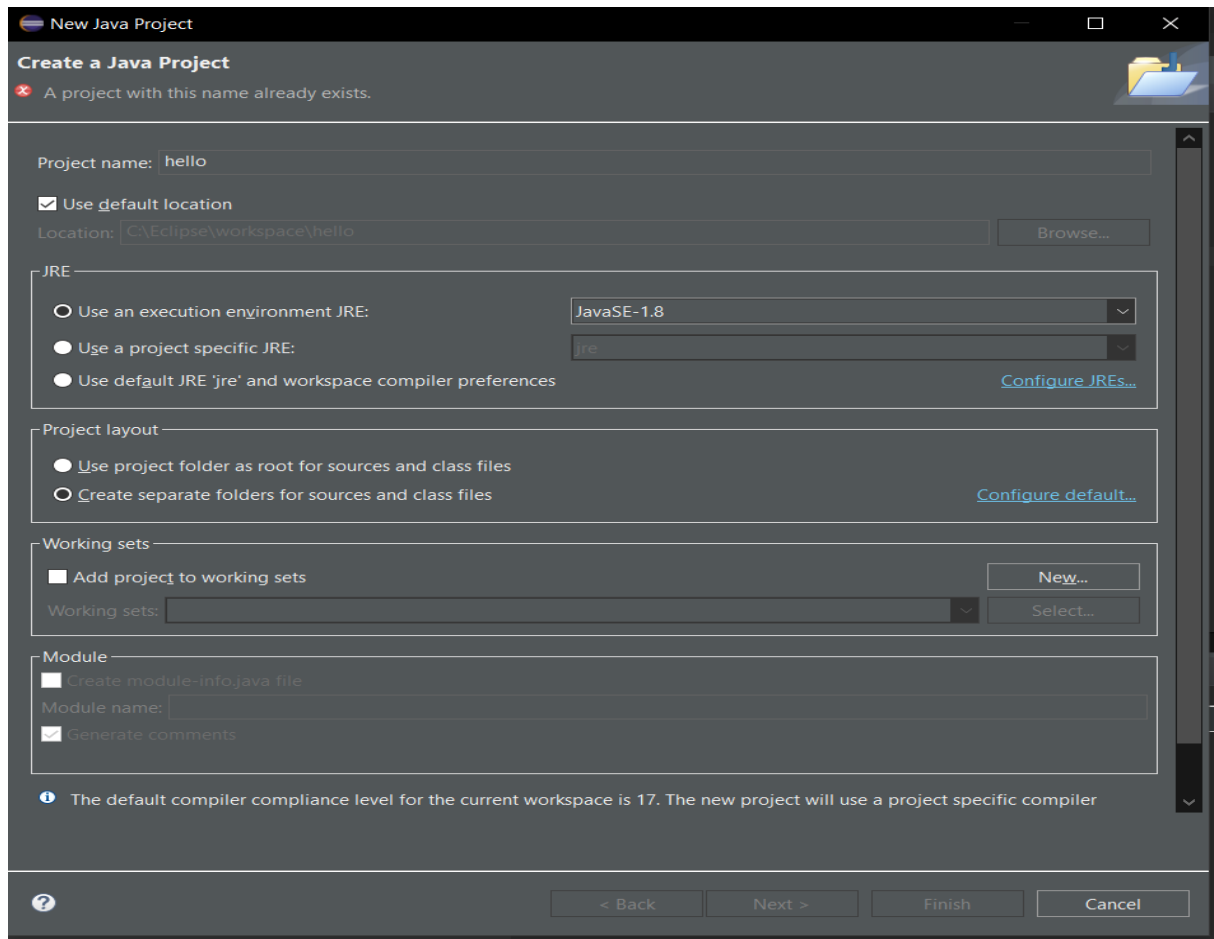
## II. Installation de l'IDE - Eclipse :



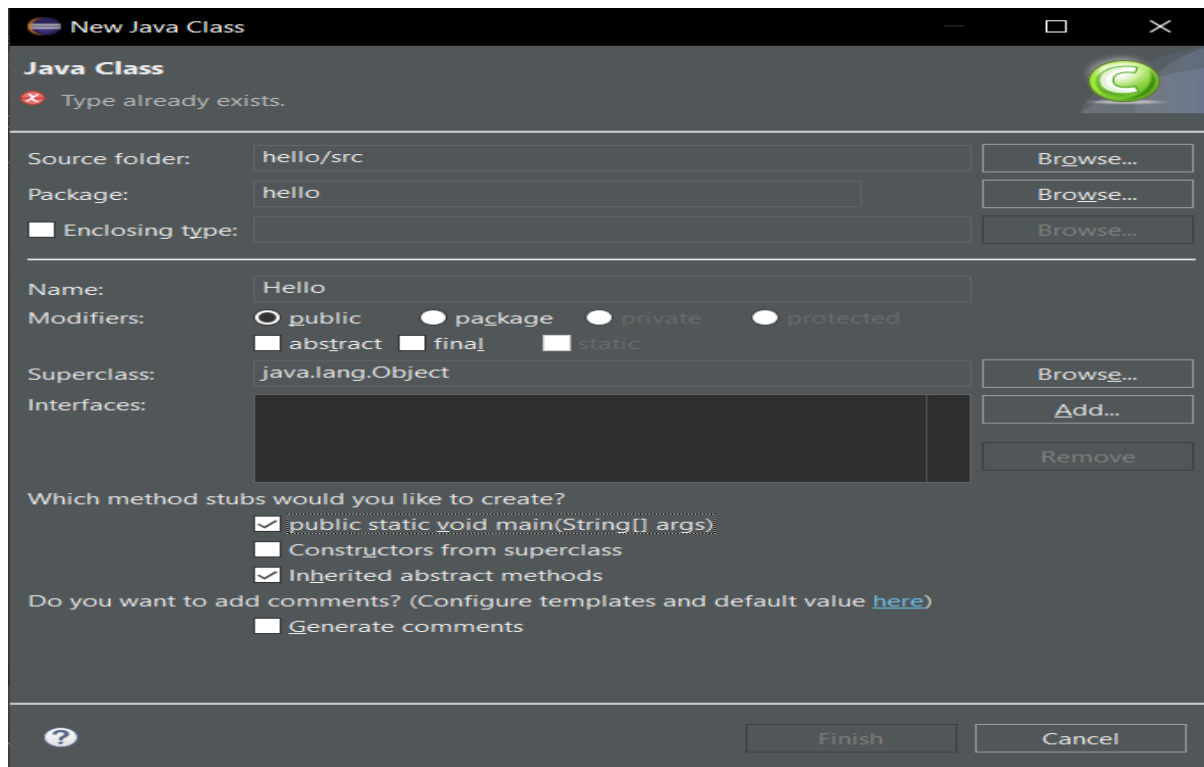
## III. L'indispensable « Hello Word »

- Création d'un nouveau projet sous Eclipse:

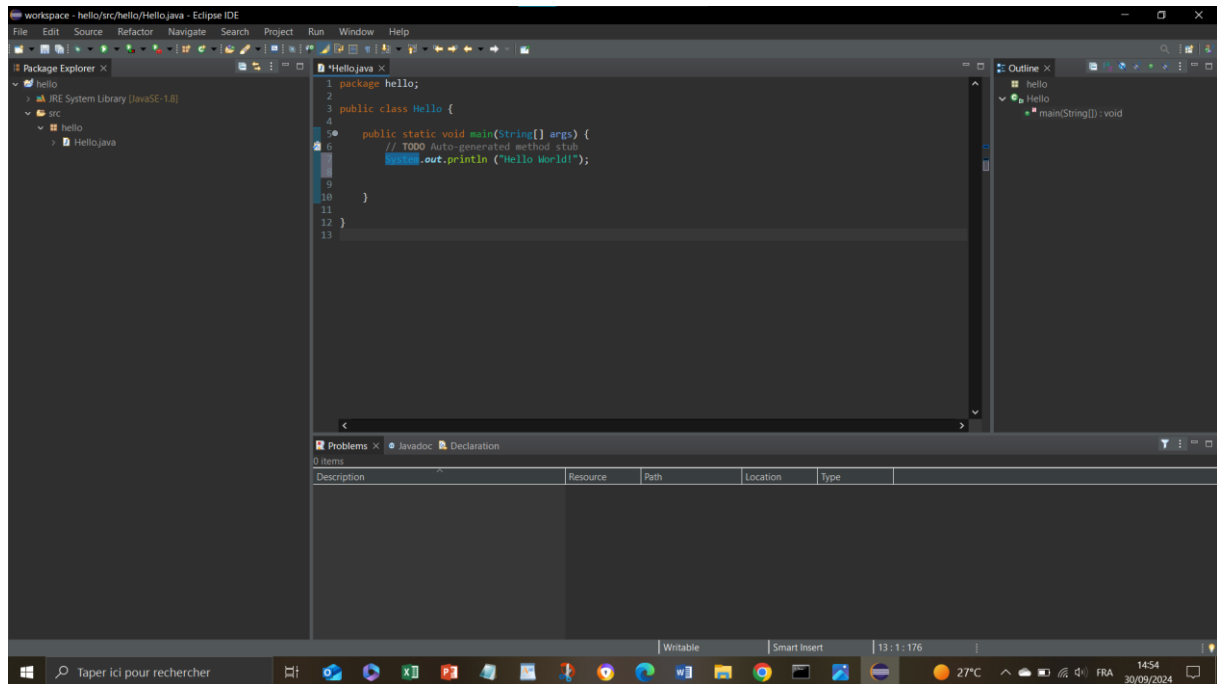




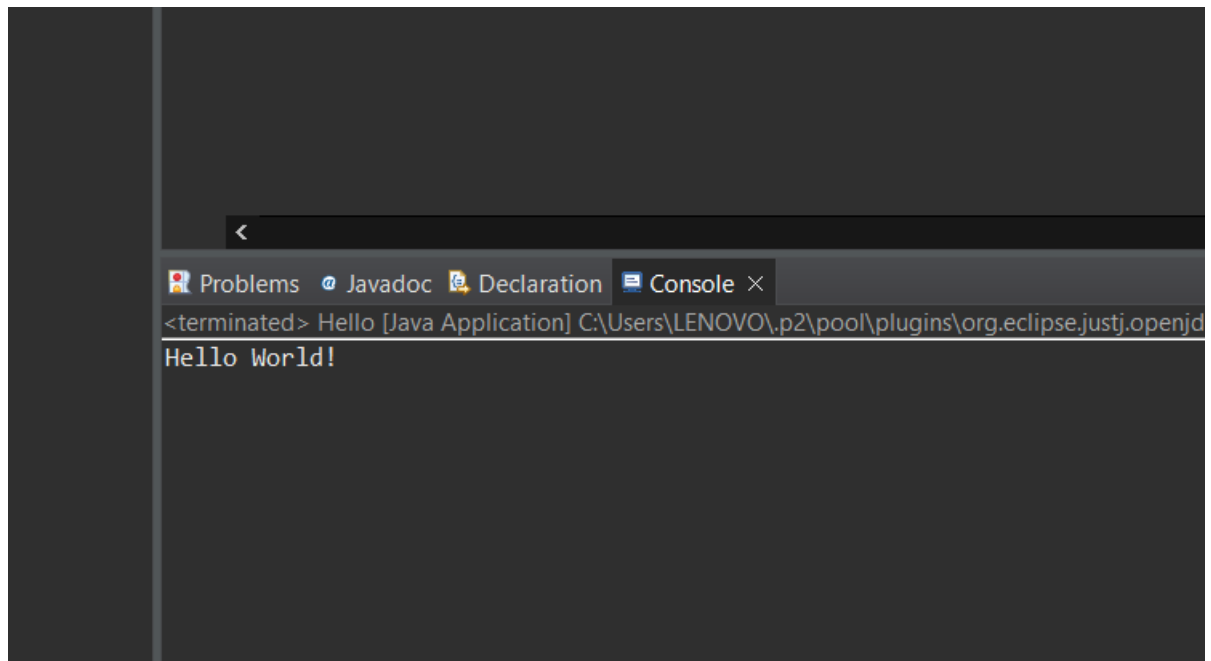
- Création d'un nouveau projet sous Eclipse:



- compléter la méthode main, en ajoutant `System.out.println ("Hello World!");`

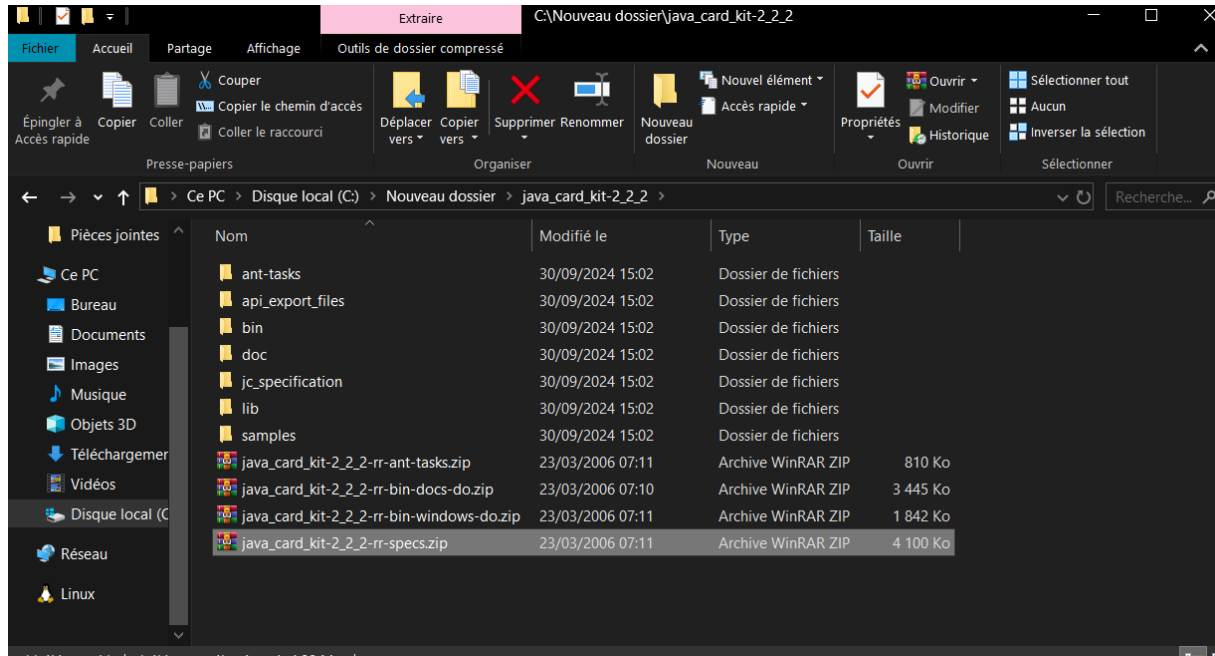


- tester notre programme,

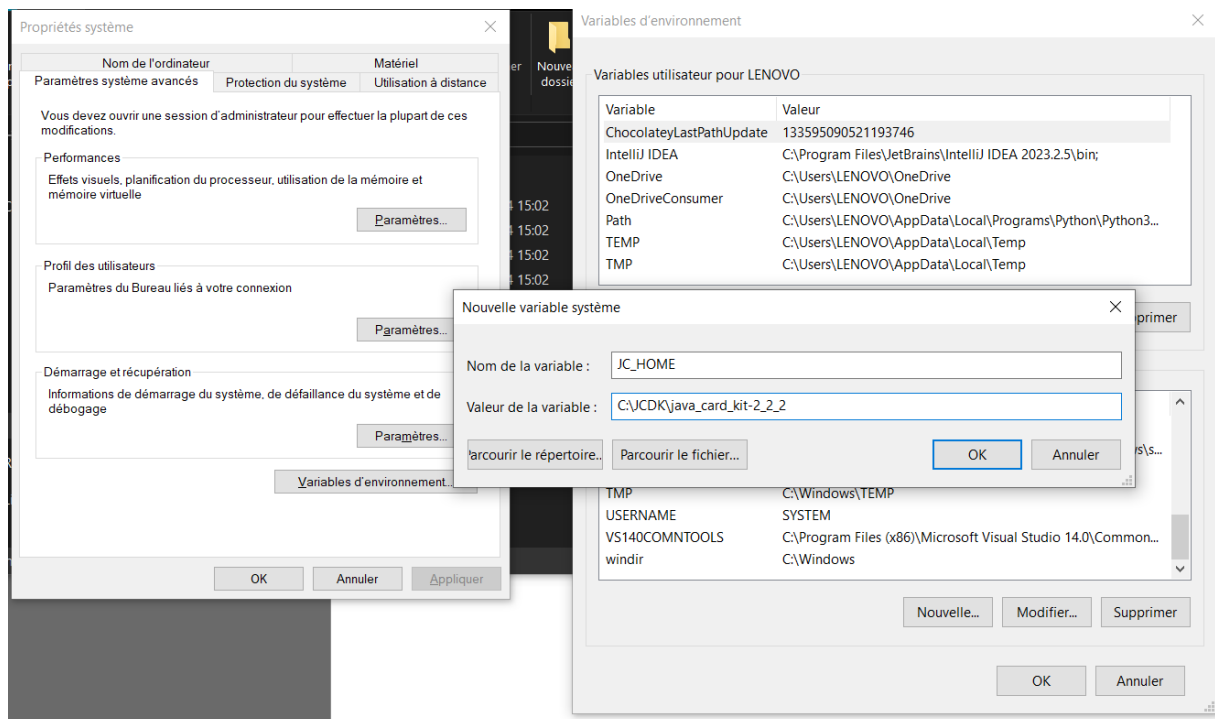


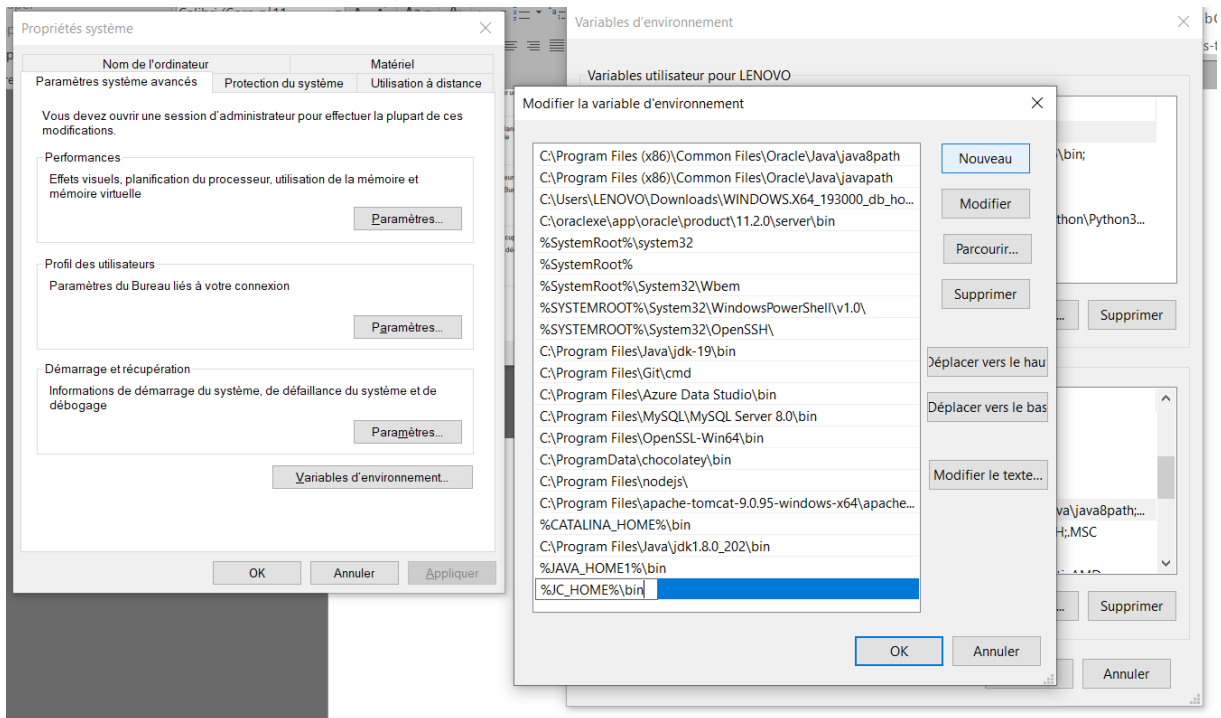


## IV. Installation du Java Card Development Kit 2.2.2 :



- ajouter la variable d'environnement JC\_HOME





# TP2 – Développement d’une application coté Serveur (V2.0)

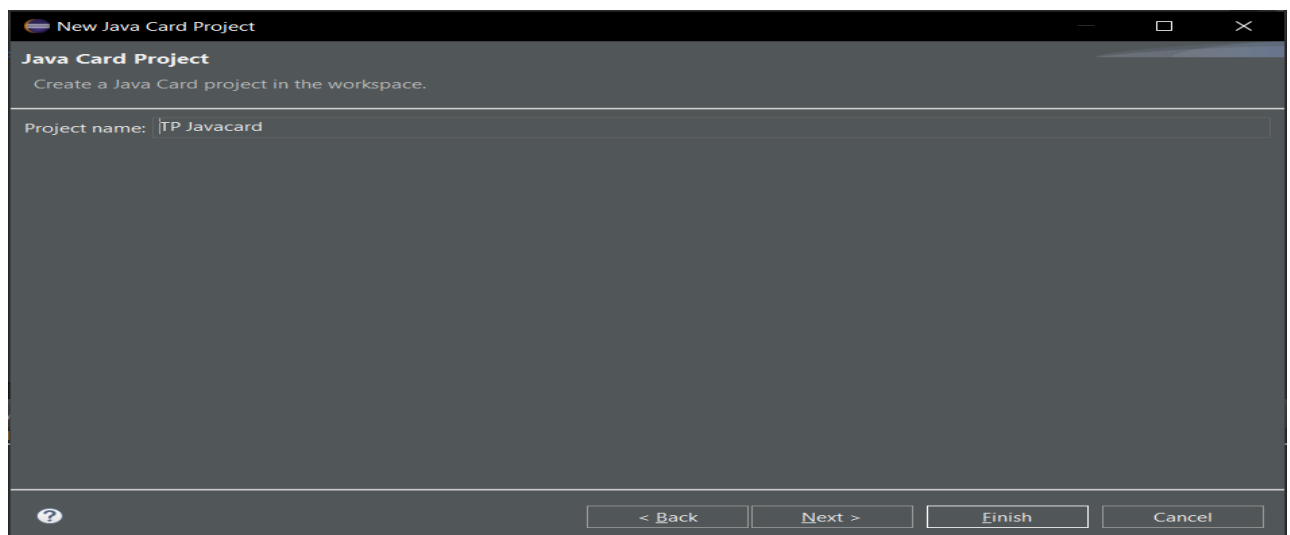
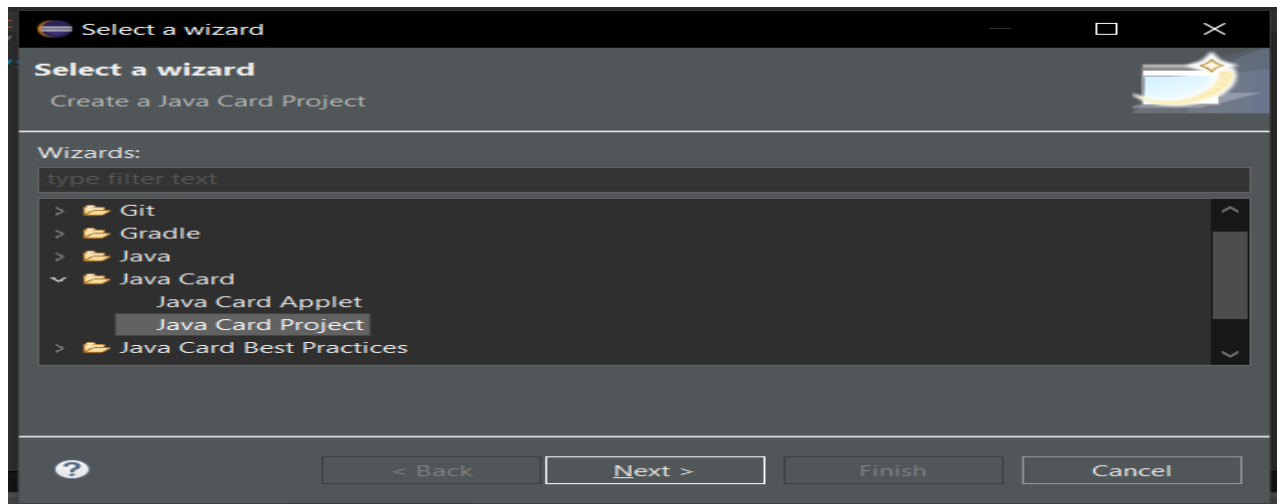
## But de ce TP

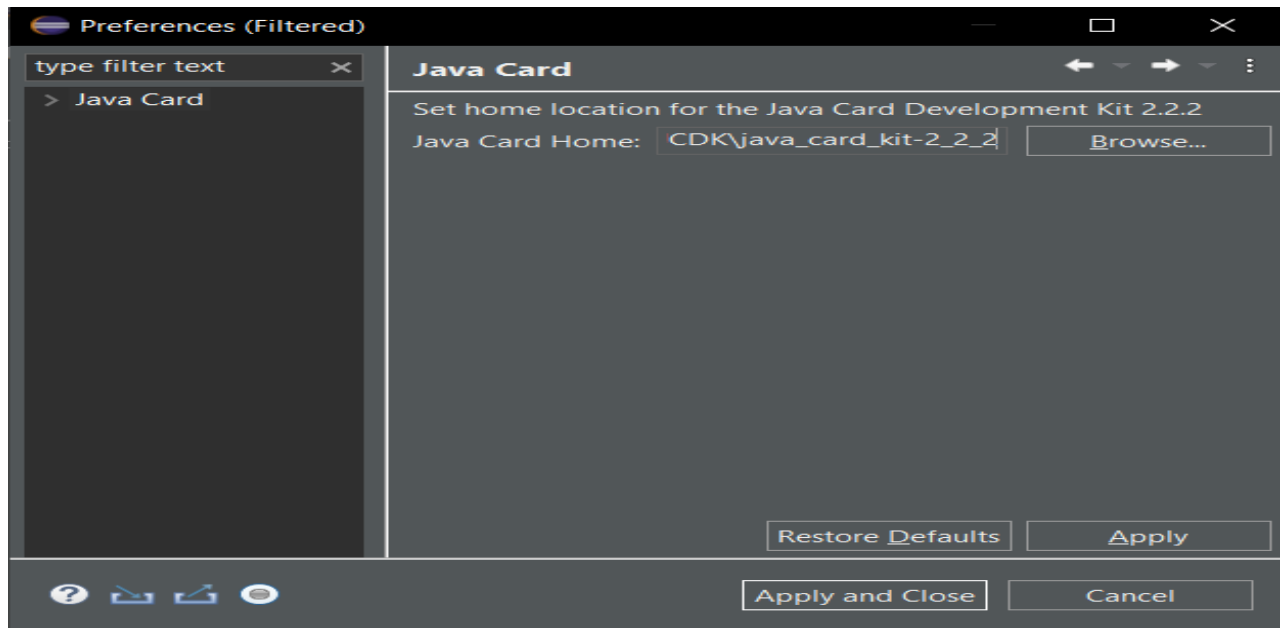
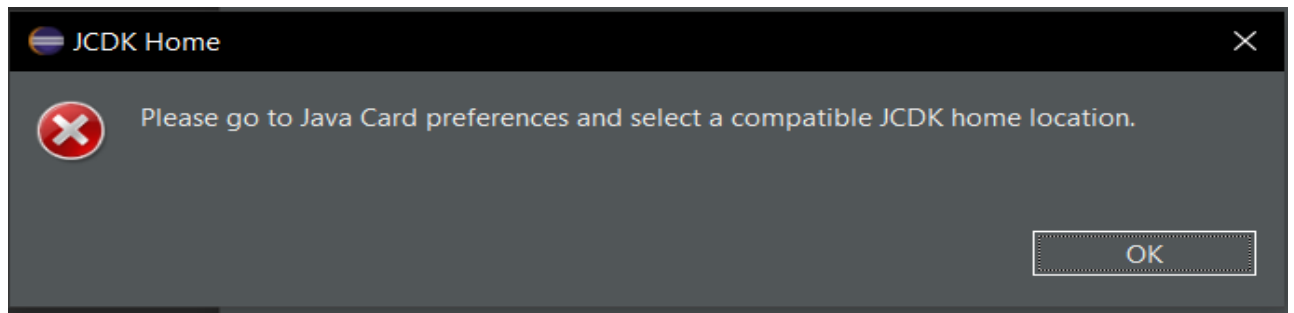
- Développer une première applet Java Card (appelée parfois cardlet car elle s'exécute sur la carte à puce).
- Tester à l'aide d'un simulateur de carte.

### I. Programmation de l'application Serveur

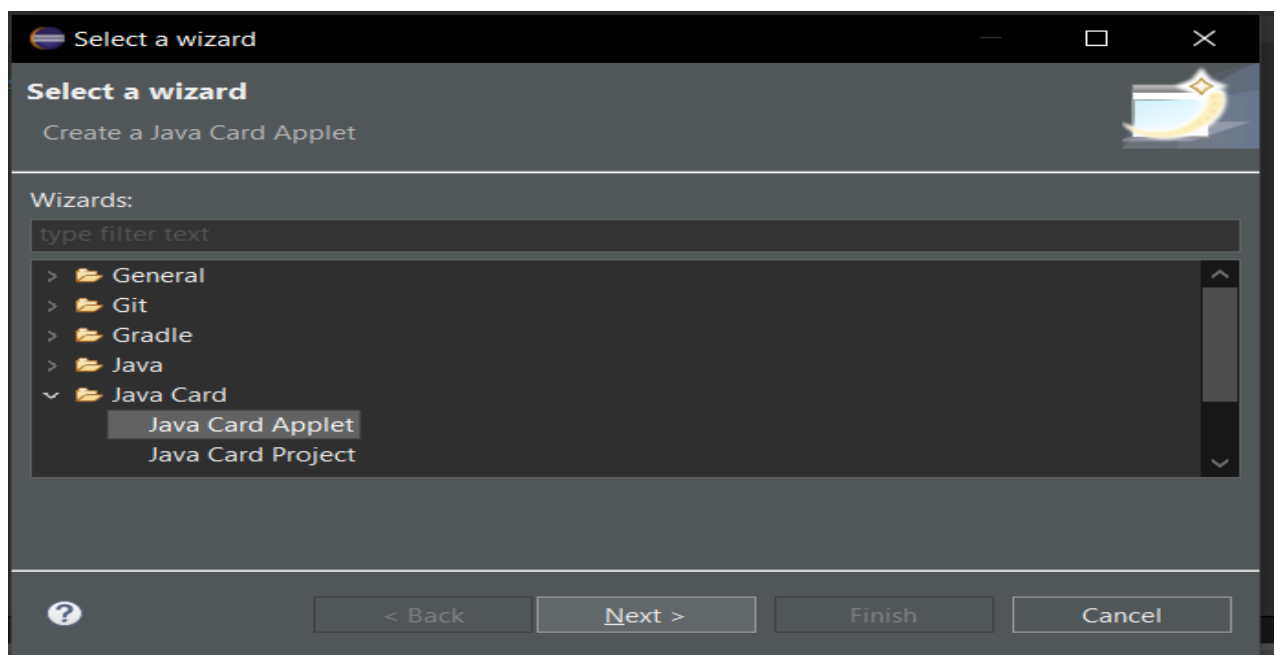
#### 1. Création de l'applet card sous Eclipse :


- Création d’un nouveau projet :





- Création d'une applet Javacard:



 New Java Card Applet

## Java Card Applet

Create a new Java Card Applet class.

Source folder:
TP Javacard/src
Browse...

Package:
monpackage
Browse...

☐ Enclosing type:
Browse...

Applet AID
0x01:0x02:0x03:0x04:0x05:0x06:0x07:0x08:0x09:0x00:0x00

Name:
MonApplet

Modifiers:

☐ public
☒ package
☐ private
☐ protected

☐ abstract
☐ final
☐ static


☐ none
☒ sealed
☐ non-sealed
☐ final

Superclass:
javacard.framework.Applet
Browse...

Interfaces:

Add...
Remove

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments



< Back

Next >

Finish

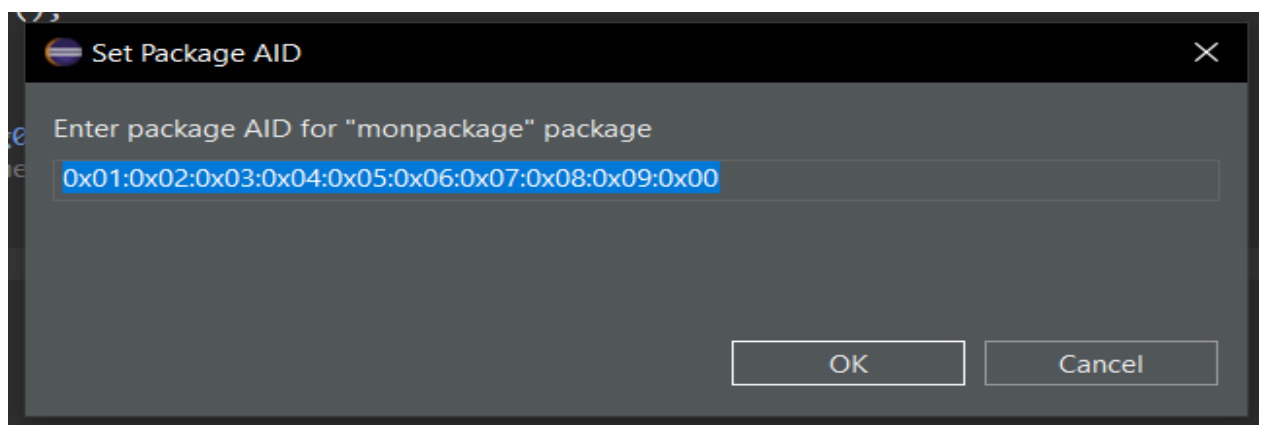
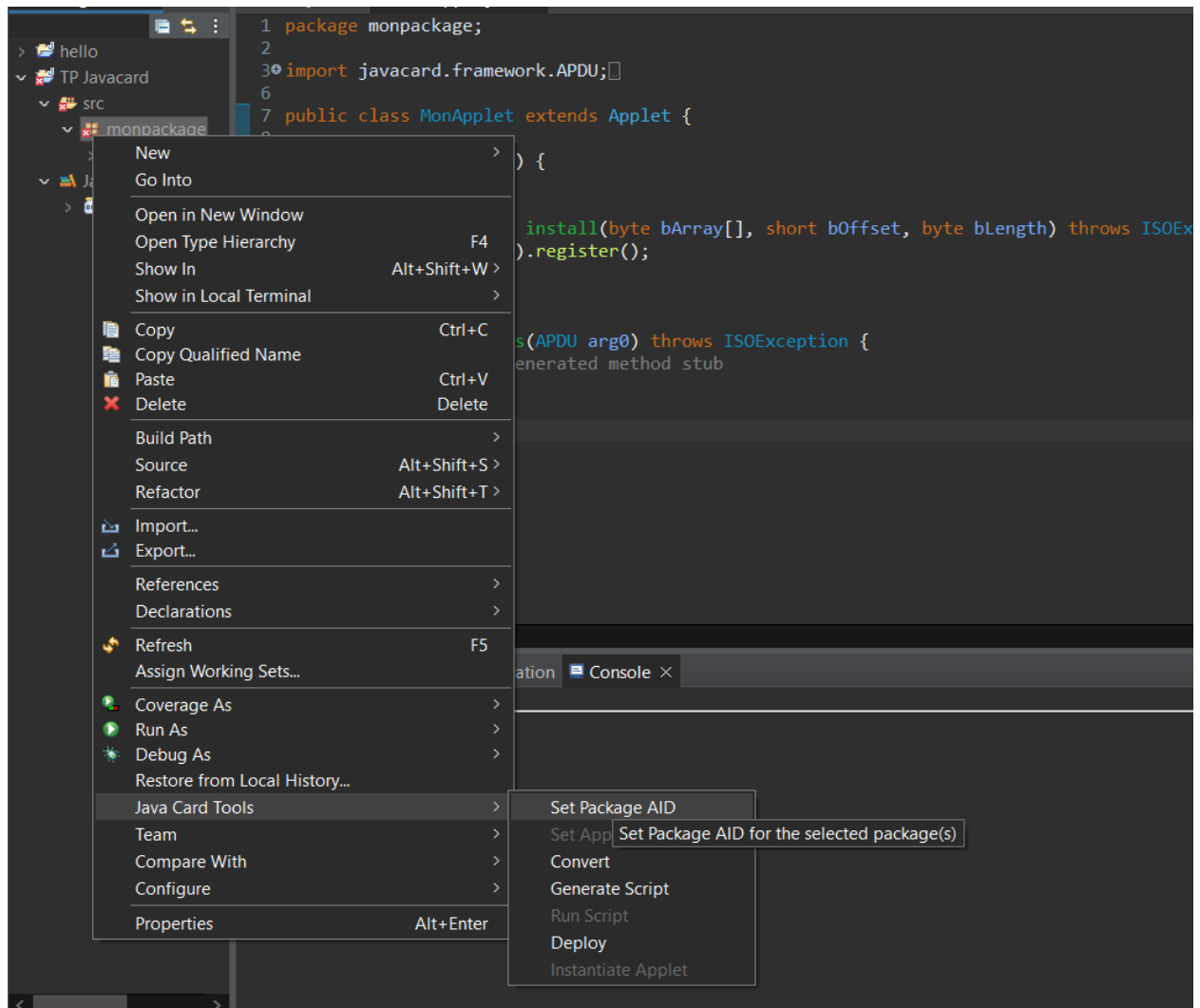
Cancel

Package Explorer
Hello.java
MonApplet.java

```

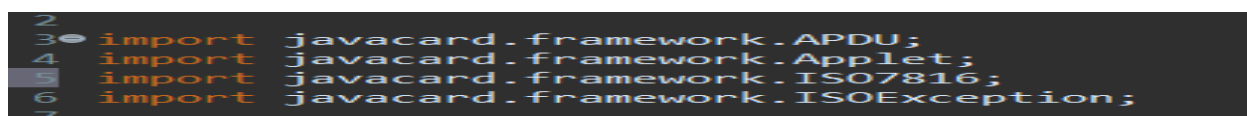
1 package monpackage;
2 hello/src/hello/Hello.java
3 import javacard.framework.APDU;
4
5
6
7 public class MonApplet extends Applet {
8
9     private MonApplet() {
10    }
11
12    public static void install(byte bArray[], short bOffset, byte bLength) throws ISOException {
13        new MonApplet().register();
14    }
15
16    @Override
17    public void process(APDU arg0) throws ISOException {
18        // TODO Auto-generated method stub
19    }
20
21
22 }
23

```



## 2. Codage de notre applet :

- Étape 1. Ajouter API JavaCard :



- Étape 2. Déclarer les attributs et les constantes :

```
public class MonApplet extends Applet {

    public static final byte CLA_MONAPPLET = (byte) 0xB0;

    public static final byte INS_INCREMENTER_COMPTEUR = 0x00;
    public static final byte INS_DECREMENTER_COMPTEUR = 0x01;
    public static final byte INS_INTERROGER_COMPTEUR = 0x02;
    public static final byte INS_INITIALISER_COMPTEUR = 0x03;

    private byte compteur;
```

- Étape 3. Définition des méthodes publiques qu'elle doit obligatoirement implémenter :

- \* La méthode install () :

```
private MonApplet() {
    compteur = 0;
}

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

- \* La méthode process () :

```
public void process(APDU apdu) throws ISOException {
    // TODO Auto-generated method stub
    byte[] buffer = apdu.getBuffer();

    if (this.selectingApplet()) return;

    if (buffer[ISO7816.OFFSET_CLA] != CLA_MONAPPLET) {
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
    }

    switch (buffer[ISO7816.OFFSET_INS]) {
        case INS_INCREMENTER_COMPTEUR:
            compteur++;
            break;

        case INS_DECREMENTER_COMPTEUR:
            compteur--;
            break;

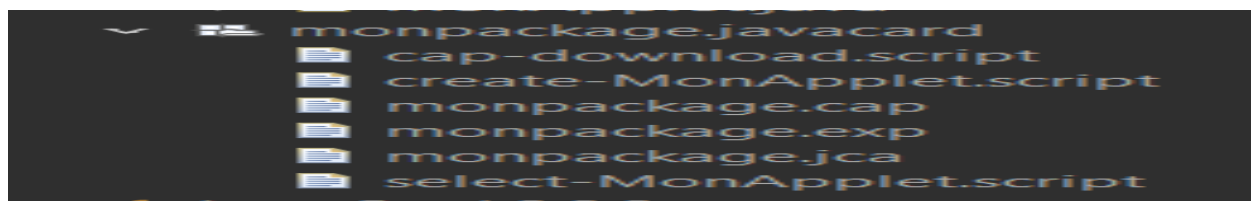
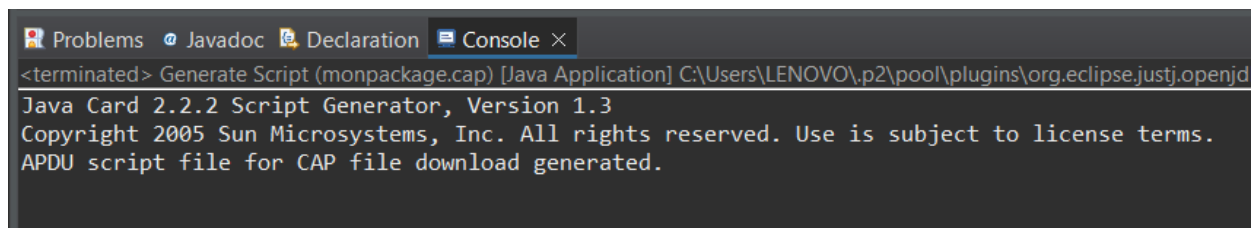
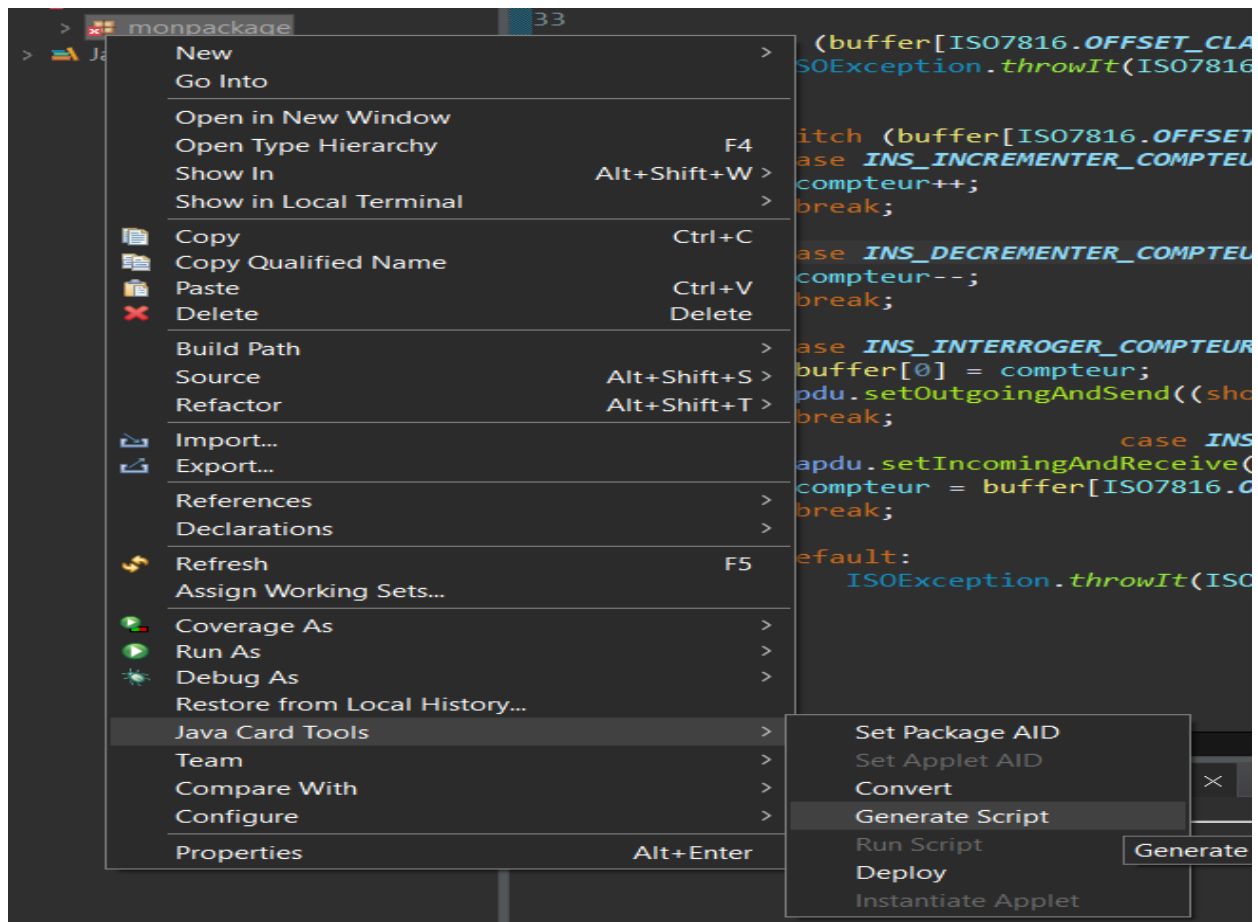
        case INS_INTERROGER_COMPTEUR:
            buffer[0] = compteur;
            apdu.setOutgoingAndSend((short) 0, (short) 1);
            break;

        case INS_INITIALISER_COMPTEUR:
            apdu.setIncomingAndReceive();
            compteur = buffer[ISO7816.OFFSET_CDATA];
            break;

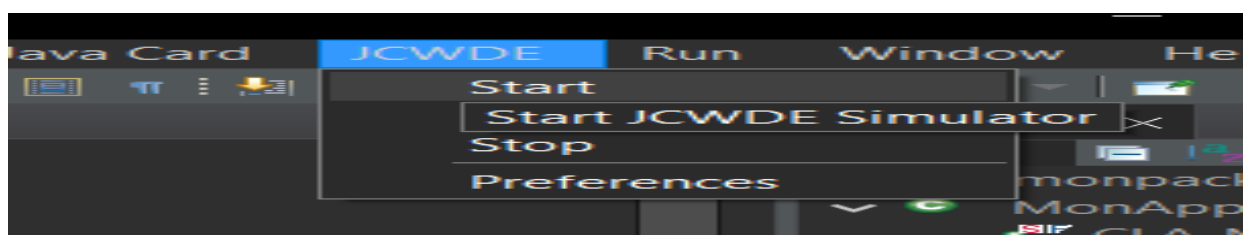
        default:
            ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }
}

}
```

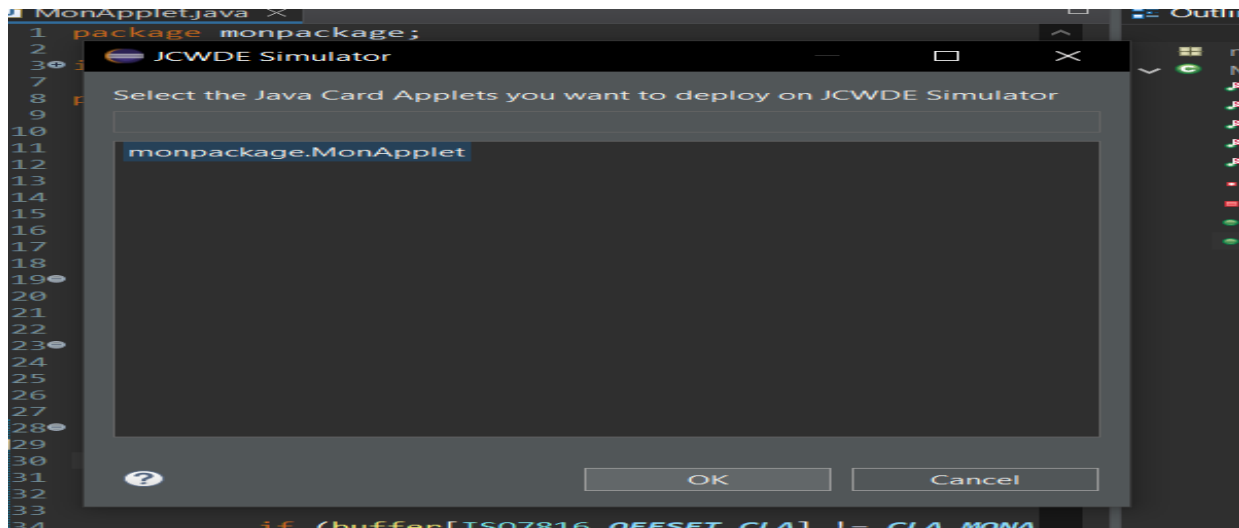
### 3. Outils de simulation :



- JCWDE : simulateur sans conservation d'état







```

C:\Users\LENOVO>setvars.bat
JAVA_HOME: C:\Program Files\Java\jdk-17
JC_HOME: C:\JCDK\java_card_kit-2_2_2

C:\Users\LENOVO>_

```

```

C:\Users\LENOVO>apdutool
Java Card 2.2.2 APDU Tool, Version 1.3
Copyright 2005 Sun Microsystems, Inc. All rights reserved. Use is subject to license terms.
Opening connection to localhost on port 9025.
Connected.

```

```

C:\Users\LENOVO>apdutool
Java Card 2.2.2 APDU Tool, Version 1.3
Copyright 2005 Sun Microsystems, Inc. All rights reserved. Use is subject to license terms.
Opening connection to localhost on port 9025.
Connected.
powerup;
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00

```

```

C:\Users\LENOVO>apdutool
Java Card 2.2.2 APDU Tool, Version 1.3
Copyright 2005 Sun Microsystems, Inc. All rights reserved. Use is subject to license terms.
Opening connection to localhost on port 9025.
Connected.
powerup;
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00
0x00 0xA4 0x04 0x00 0x09 0xA0 0x00 0x00 0x00 0x62 0x03 0x01 0x08 0x01 0x7F;
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 09, a0, 00, 00, 00, 62, 03, 01, 08, 01, Le: 00, SW1: 90, SW2: 00
0x80 0xB8 0x00 0x00 0xd 0xb 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x00 0x00 0x7F;
CLA: 80, INS: b8, P1: 00, P2: 00, Lc: 0d, 0b, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, 00, 00, Le: 0b, 01
, 02, 03, 04, 05, 06, 07, 08, 09, 00, 00, SW1: 90, SW2: 00
_

```

```

powerup;
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00
0x00 0xA4 0x04 0x00 0xb 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x00 0x7F;
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 0b, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, 00, Le: 00, SW1: 90, S
W2: 00

```

```

0xB0 0x02 0x00 0x00 0x00 0x7F;
CLA: b0, INS: 02, P1: 00, P2: 00, Lc: 00, Le: 01, 00, SW1: 90, SW2: 00

```

```

0xB0 0x00 0x00 0x00 0x00 0x7F;
CLA: b0, INS: 00, P1: 00, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00

```

```
0xB0 0x02 0x00 0x00 0x00 0x7F;
CLA: b0, INS: 02, P1: 00, P2: 00, Lc: 00, Le: 01, 01, SW1: 90, SW2: 00
```

```
0xB0 0x03 0x00 0x00 0x01 0x4A 0x7F;
CLA: b0, INS: 03, P1: 00, P2: 00, Lc: 01, 4a, Le: 00, SW1: 90, SW2: 00
```

```
0xB0 0x01 0x00 0x00 0x00 0x7F;
CLA: b0, INS: 01, P1: 00, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
```

```
0xB0 0x02 0x00 0x00 0x00 0x7F;
CLA: b0, INS: 02, P1: 00, P2: 00, Lc: 00, Le: 01, 49, SW1: 90, SW2: 00
```

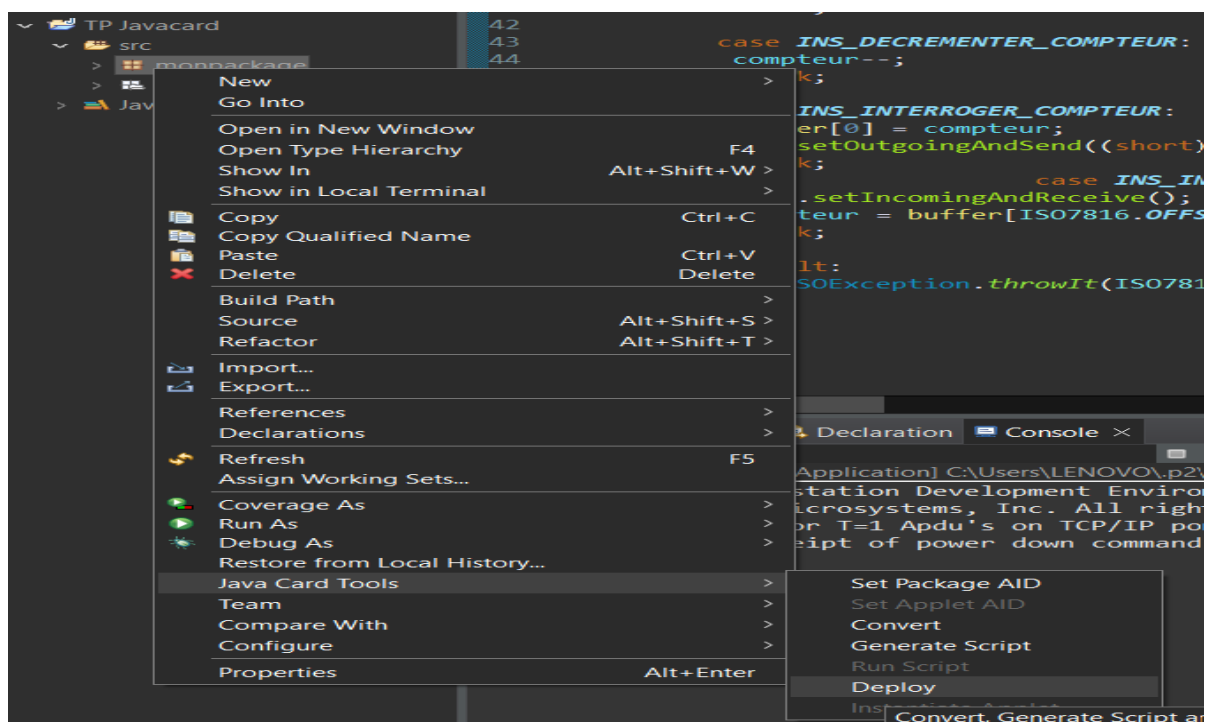
```
powerdown;
```

- CREF : simulateur avec conservation d'état

```
C:\>cref -o monapplet.eeprom
Java Card 2.2.2 C Reference Implementation Simulator (version 0.41)
32-bit Address Space implementation - with cryptography support
T=1 / T=CL Dual interface APDU protocol (ISO 7816-3)
Copyright 2005 Sun Microsystems, Inc. All rights reserved.

Memory configuration
  Type      Base      Size      Max Addr
  RAM       0x0       0x1000   0xffff
  ROM       0x2000   0xe000   0xfffff
  E2P       0x10020  0xffe0   0x1ffff

  ROM Mask size =                0xce64 =      52836 bytes
  Highest ROM address in mask =   0xee63 =     61027 bytes
  Space available in ROM =        0x119c =      4508 bytes
EEPROM will be saved in file "monapplet.eeprom"
Mask has now been initialized for use
```



```
Problems Javadoc Declaration Console X
C:\Users\LENOVO\p2\pool\plugins\org.eclipse.justi.openjdk hotspot.jre.full.win32.x86_64.17.0.12.v20240802-1518\jre\bin\javaw.exe (10 nov. 2024, 21:21:51) [pid: 8848]
Opening connection to localhost on port 30222
Connected.
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x01
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 09, a0, 00, 00, 00, 62, 03, 01, 08, 01, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b0, P1: 00, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b2, P1: 01, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 01, P2: 00, Lc: 17, 01, 00, 14, de, ca, ff, ed, 01, 02, 04, 00, 01, 0a, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: bc, P1: 01, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b2, P1: 02, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 02, P2: 00, Lc: 20, 02, 00, 1f, 00, 14, 00, 1f, 00, 0f, 00, 15, 00, 2a, 00, 0c, 00, 7d, 00, 0a, 00, 15, 00, 00, 00, 61, 00, 00, 00,
CLA: 80, INS: b4, P1: 02, P2: 00, Lc: 02, 01, 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: bc, P1: 02, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b2, P1: 04, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 04, P2: 00, Lc: 18, 04, 00, 15, 02, 03, 01, 07, a0, 00, 00, 00, 62, 01, 01, 00, 01, 07, a0, 00, 00, 00, 62, 00, 01, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: bc, P1: 04, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b2, P1: 03, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 03, P2: 00, Lc: 12, 03, 00, 0f, 01, 0b, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, 00, 00, 0c, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: bc, P1: 03, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b2, P1: 06, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 06, P2: 00, Lc: 0f, 06, 00, 0c, 00, 80, 03, 01, ff, 00, 07, 01, 00, 00, 00, 19, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: bc, P1: 06, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b2, P1: 07, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 07, P2: 00, Lc: 20, 07, 00, 7d, 00, 02, 10, 18, 8c, 00, 01, 18, 03, 88, 00, 7a, 02, 30, 8f, 00, 02, 3d, 8c, 00, 03, 8b, 00, 04, 7a,
CLA: 80, INS: b4, P1: 07, P2: 00, Lc: 20, 00, 05, 2d, 18, 8b, 00, 06, 60, 03, 7a, 1a, 03, 25, 10, b0, 6a, 08, 11, 6e, 00, 8d, 00, 07, 1a, 04, 25, 73, 00,
CLA: 80, INS: b4, P1: 07, P2: 00, Lc: 20, 03, 00, 0f, 00, 1a, 00, 25, 00, 32, 18, 3d, 84, 00, 04, 41, 5b, 88, 00, 70, 2d, 18, 3d, 84, 00, 04, 43, 5b, 88,
CLA: 80, INS: b4, P1: 07, P2: 00, Lc: 20, 03, ae, 00, 38, 19, 03, 04, 8b, 00, 08, 70, 15, 19, 8b, 00, 09, 3b, 18, 1a, 08, 25, 88, 00, 70, 08, 11, 6d, 00,
CLA: 80, INS: bc, P1: 07, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b2, P1: 08, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 08, P2: 00, Lc: 0d, 08, 00, 0a, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: bc, P1: 08, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b2, P1: 05, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 05, P2: 00, Lc: 20, 05, 00, 2a, 00, 0a, 02, 00, 00, 00, 06, 80, 03, 00, 01, 00, 00, 00, 06, 00, 00, 01, 03, 80, 03, 01, 03, 80, 0a,
CLA: 80, INS: b4, P1: 05, P2: 00, Lc: 0d, 03, 06, 80, 07, 01, 03, 80, 0a, 08, 03, 80, 0a, 06, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: bc, P1: 05, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b2, P1: 09, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b4, P1: 09, P2: 00, Lc: 18, 09, 00, 15, 00, 07, 0a, 3f, 05, 06, 05, 06, 14, 00, 0a, 05, 0a, 04, 03, 07, 05, 10, 33, 06, 0f, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: bc, P1: 09, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: ba, P1: 00, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
```

```
C:\>crcf -i monapplet.eeprom -o monapplet.eeprom
Java Card 2.2.2 C Reference Implementation Simulator (version 0.41)
32-bit Address Space implementation - with cryptography support
T=1 / T=CL Dual interface APDU protocol (ISO 7816-3)
Copyright 2005 Sun Microsystems, Inc. All rights reserved.

Memory configuration
Type      Base      Size      Max Addr
RAM       0x0       0x1000    0xffff
ROM       0x2000    0xe000    0xfffff
E2P       0x10020   0xffe0    0x1ffff

ROM Mask size =          0xce64 =          52836 bytes
Highest ROM address in mask = 0xee63 =          61027 bytes
Space available in ROM =  0x119c =           4508 bytes
EEPROM (0xfffe0 bytes) restored from file "monapplet.eeprom"
Using a pre-initialized Mask
```

```
TP Javacard
src
monpackage
monpackage.javacard
cap-download.script
create-MonApplet.script
New
Open
Open With
Show In
Show in Local Terminal
Copy
Copy Qualified Name
Paste
Delete
Build Path
Refactor
Import...
Export...
Refresh
Assign Working Sets...
Coverage As
Run As
Debug As
Java Card Tools
Team
Compare With
Replace With
Properties
Alt+Enter
// Auto-generated method stub
byte[] buffer = apdu.getBuffer();
if (this.selectingApplet()) return;
if (buffer[ISO7816.OFFSET_CLA] != CLA_SELECT)
    ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
buffer[ISO7816.OFFSET_INS] = INS_INITIATE;
INCREMENTER_COMPTEUR++;
DECREMENTER_COMPTEUR--;
INTERROGER_COMPTEUR++;
OutgoingAndSend((short) 0, buffer.length);
case INS_INITIATE:
IncomingAndReceive();
buffer[ISO7816.OFFSET_CLA] = CLA_SELECT;
ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
Set Package AID
Set Applet AID
Convert
Generate Script
Run Script
Deploy
InstantiateApplet
Run the selected APDU Script
```

```
Java Card 2.2.2 APDU Tool, Version 1.3
Copyright 2005 Sun Microsystems, Inc. All rights reserved. Use is subject to license terms.
Opening connection to localhost on port 9025.
Connected.
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x01
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 09, a0, 00, 00, 00, 62, 03, 01, 08, 01, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b8, P1: 00, P2: 00, Lc: 0d, 0b, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, 00, 00, Le: 0b, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, 00, SW1: 90, SW2: 00
|
```

```
C:\>apdutool
Java Card 2.2.2 APDU Tool, Version 1.3
Copyright 2005 Sun Microsystems, Inc. All rights reserved. Use is subject to license terms.
Opening connection to localhost on port 9025.
Connected.
powerup;
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x01
0x00 0xA4 0x04 0x00 0x0b 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x00 0x7F;
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 0b, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, 00, Le: 00, SW1: 6d, S
W2: 00
```

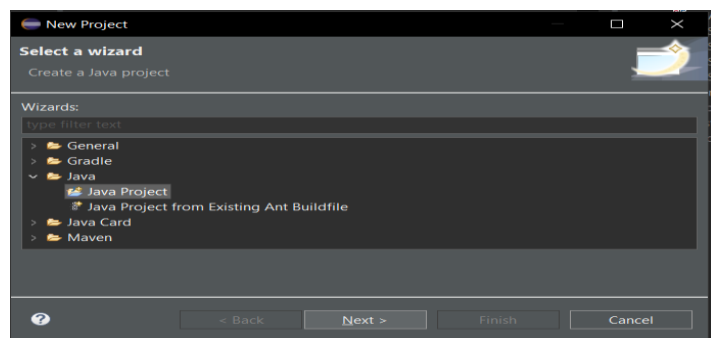
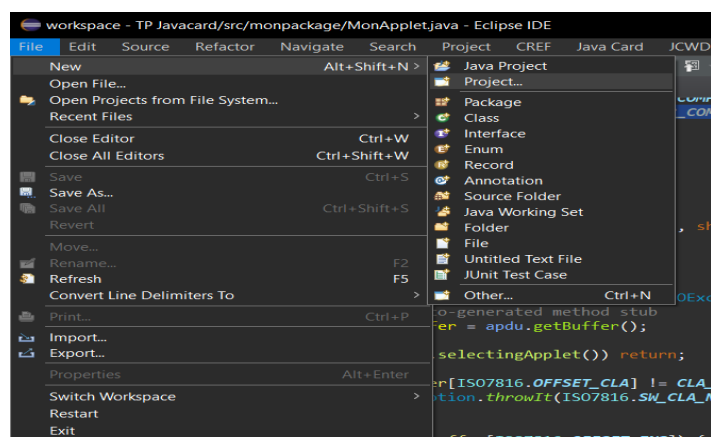
# TP3 - Programmation d'une application coté client

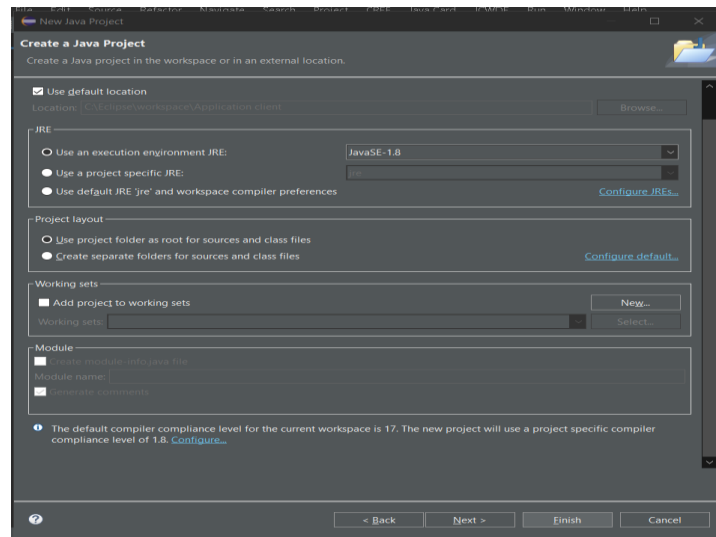
## But de ce TP

Coder une application cliente : l'équivalent du terminal bancaire si notre Javacard était une carte de paiement.

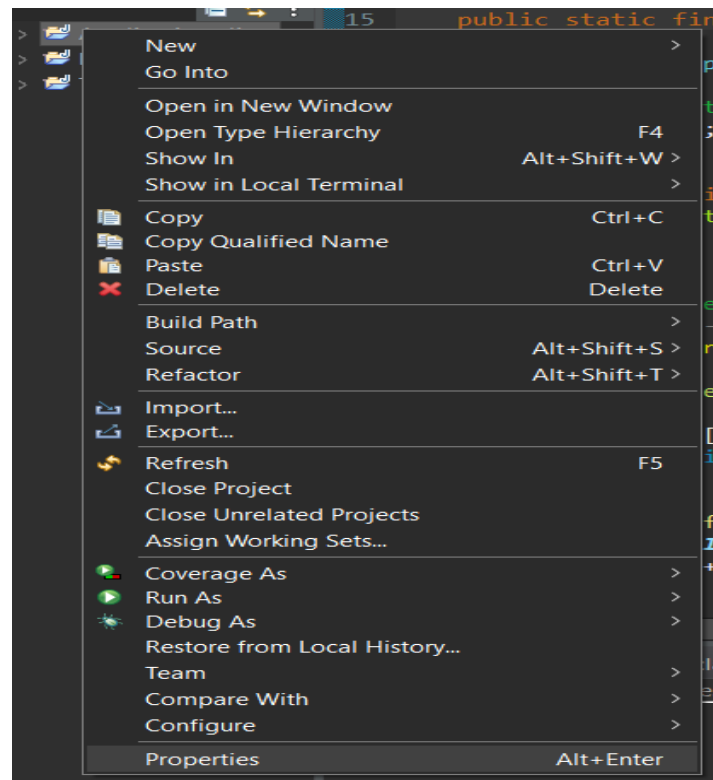
## I. Création de l'application client sous Eclipse

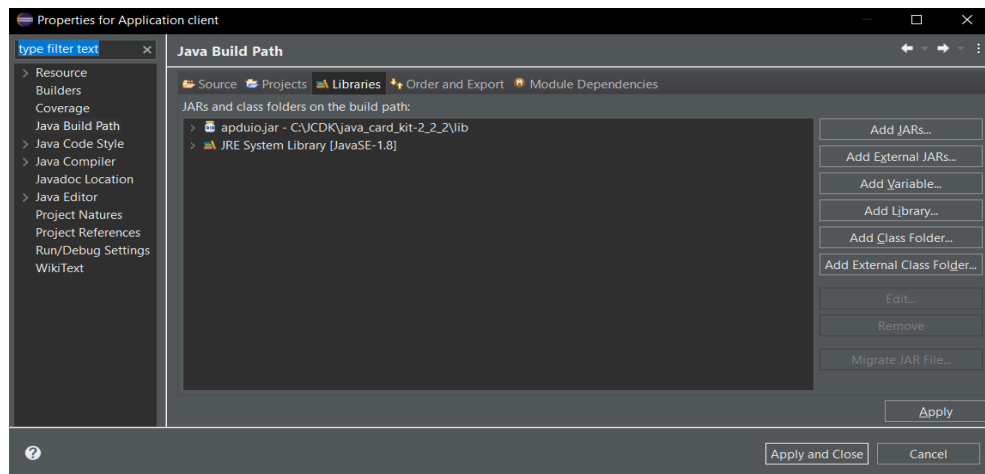
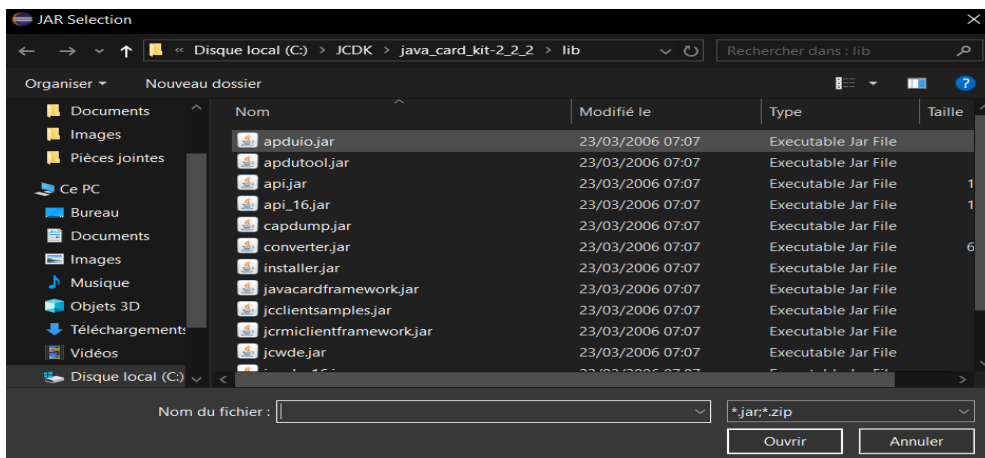
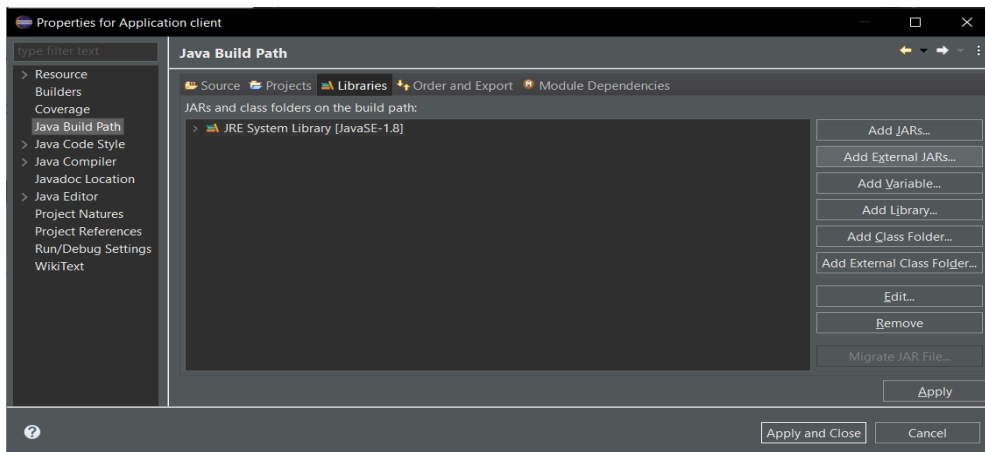
### 1. Création d'un nouveau projet



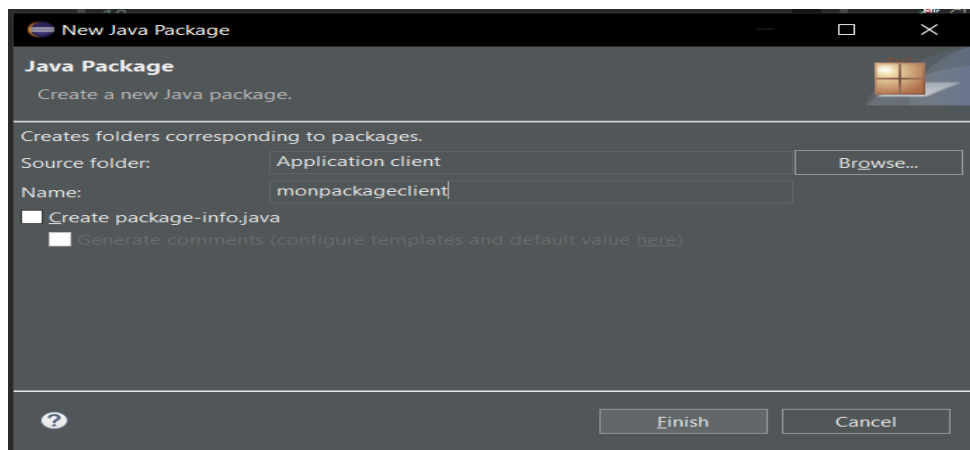
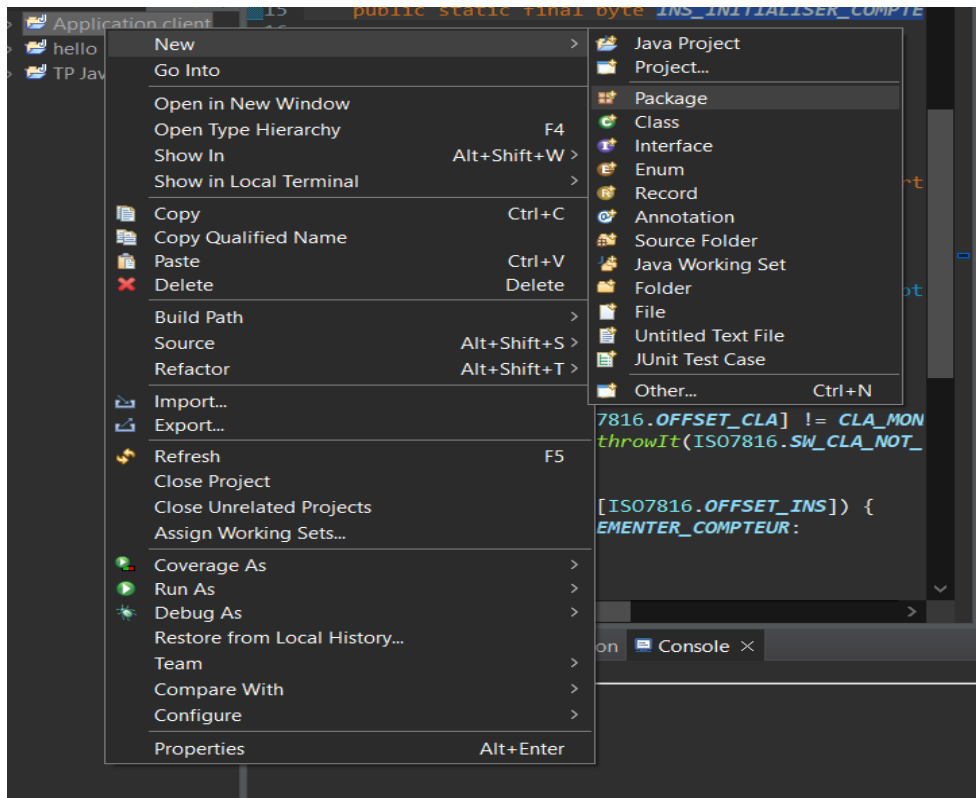


## 2. Ajout de la librairie « apduio » dans le classpath

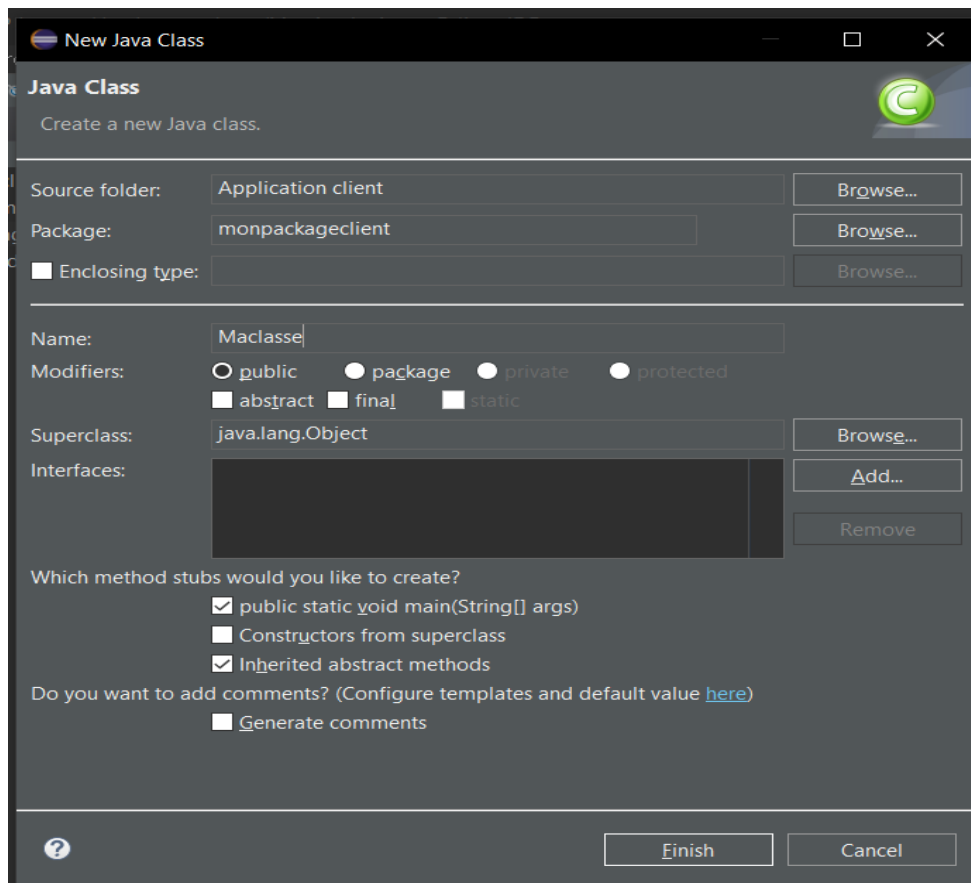
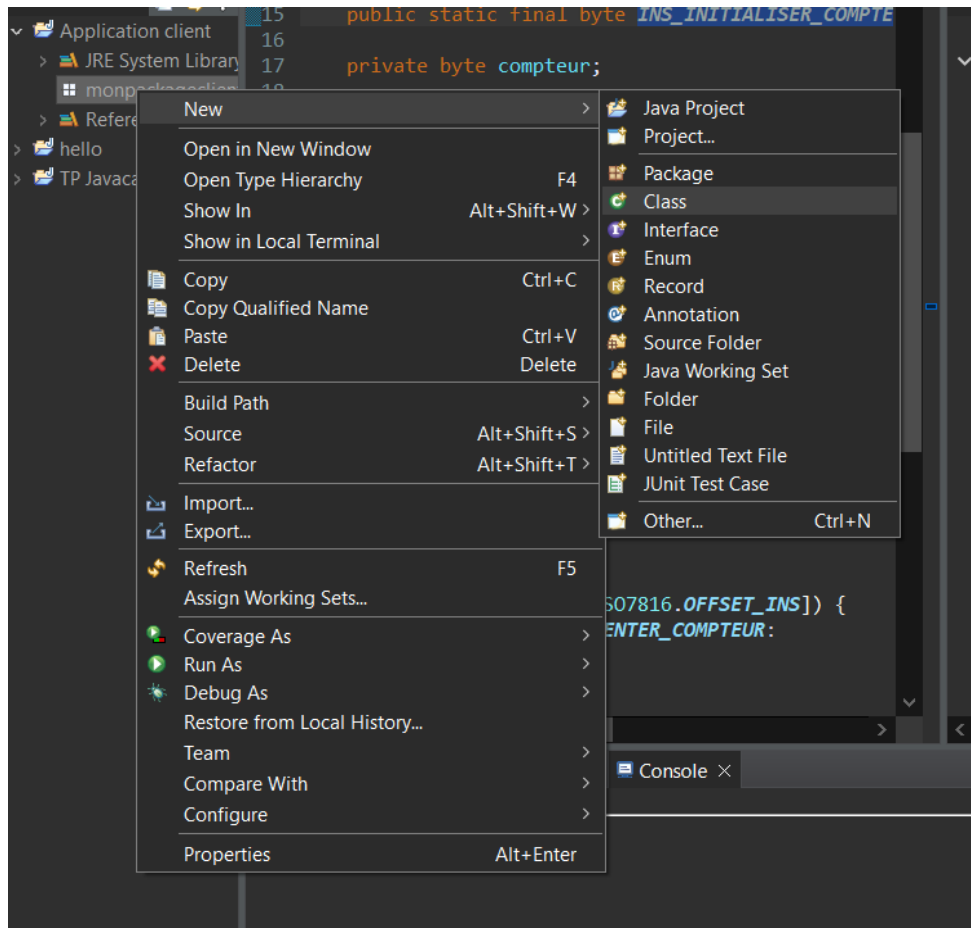




### 3. Création de la classe principale







- **Étape 1 - Connexion :**

```
public static void main(String[] args) {
    /* Connexion - Javacard */
    CadT1Client cad;
    Socket sckCarte;

    try {
        sckCarte = new Socket("localhost", 9025);
        sckCarte.setTcpNoDelay(true);
        BufferedInputStream input = new BufferedInputStream(sckCarte.getInputStream());
        BufferedOutputStream output = new BufferedOutputStream(sckCarte.getOutputStream());
        cad = new CadT1Client(input, output);
    } catch (IOException e) {
        System.out.println("Erreur : impossible de se connecter à la Javacard");
        return;
    }

    /* Mise sous tension de la carte */
    try {
        cad.powerUp();
    } catch (IOException | CadTransportException e) {
        System.out.println("Erreur lors de l'envoi de la commande Powerup à la Javacard");
        return;
    }
}
```

- **Étape 2 – Sélection :**

```
/* Sélection de l'applet */
Apu apdu = new Apdu();
apdu.command[Apu.CLA] = 0x00;
apdu.command[Apu.INS] = (byte) 0xA4;
apdu.command[Apu.P1] = 0x04;
apdu.command[Apu.P2] = 0x00;
byte[] appletAID = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x00, 0x00 };
apdu.setDataIn(appletAID);
try {
    cad.exchangeApu(apdu);
} catch (IOException | CadTransportException e) {
    System.out.println("Erreur lors de la sélection de l'applet");
    System.exit(1);
}

if (apdu.getStatus() != 0x9000) {
    System.out.println("Erreur lors de la sélection de l'applet");
    System.exit(1);
}
```

- **Étape 3 - Invocation des services implémentés :**

```
boolean fin = false;
while (!fin) {
    System.out.println();
    System.out.println("Application cliente Javacard");
    System.out.println("-----");
    System.out.println("1 - Interroger le compteur");
    System.out.println("2 - Incrémenter le compteur");
    System.out.println("3 - Décrémenter le compteur");
    System.out.println("4 - Réinitialiser le compteur");
    System.out.println("5 - Quitter");
    System.out.println();
    System.out.print("Votre choix ?");

    int choix = System.in.read();
    while (!(choix >= '1' && choix <= '5')) {
        choix = System.in.read();
    }

    Apdu apdu = new Apdu();
    apdu.command[Apu.CLA] = Maclasse.CLA_MONAPPUET;
    apdu.command[Apu.P1] = 0x00;
    apdu.command[Apu.P2] = 0x00;
    apdu.setLe(0x7f);

    switch (choix) {
        case '1':
            apdu.command[Apu.INS] = Maclasse.INS_INTERROGER_COMPTeur;
            cad.exchangeApu(apdu);
            if (apdu.getStatus() != 0x9000) {
                System.out.println("Erreur : status word différent de 0x9000");
            } else {
                System.out.println("Valeur du compteur : " + apdu.dataOut[0]);
            }
            break;
    }
}
```

```

        case '2':
            apdu.command[Apdu.INS] = INS_INCREMENTER_COMPTEUR;
            cad.exchangeApdu(apdu);
            if (apdu.getStatus() != 0x9000) {
                System.out.println("Erreur : status word different de 0x9000");
            } else {
                System.out.println("OK");
            }
            break;
        case '3':
            apdu.command[Apdu.INS] = Maclasse.INS_DECREMENTER_COMPTEUR;
            cad.exchangeApdu(apdu);
            if (apdu.getStatus() != 0x9000) {
                System.out.println("Erreur : status word different de 0x9000");
            } else {
                System.out.println("OK");
            }
            break;
        case '4':
            apdu.command[Apdu.INS] = Maclasse.INS_INITIALISER_COMPTEUR;
            byte[] donnees = new byte[1];
            donnees[0] = 0;
            apdu.setDataIn(donnees);
            cad.exchangeApdu(apdu);
            if (apdu.getStatus() != 0x9000) {
                System.out.println("Erreur : status word different de 0x9000");
            } else {
                System.out.println("OK");
            }
            break;
        case '5':
            fin = true;
            break;
    }
}

```

- Étape 4 - Mise hors tension :

```

try {
    cad.powerDown();
} catch (Exception e) {
    System.out.println("Erreur lors de l'envoi de la commande Powerdown à la Javacard");
    return;
}

```

## II. Utilisation de l'application cliente avec un simulateur – JCWDE

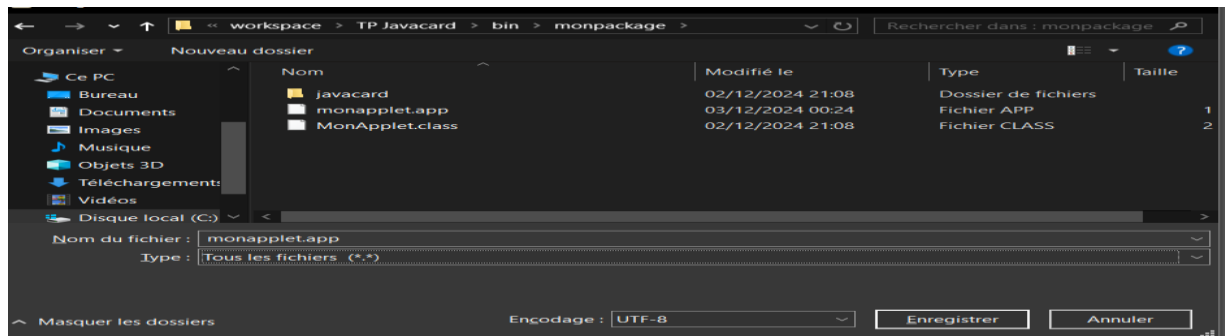
- Créer un fichier de configuration



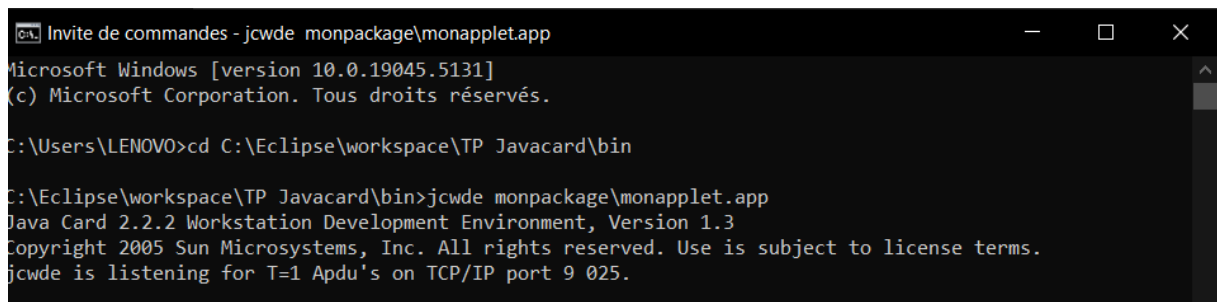
\*Sans titre - Bloc-notes

Fichier Edition Format Affichage Aide

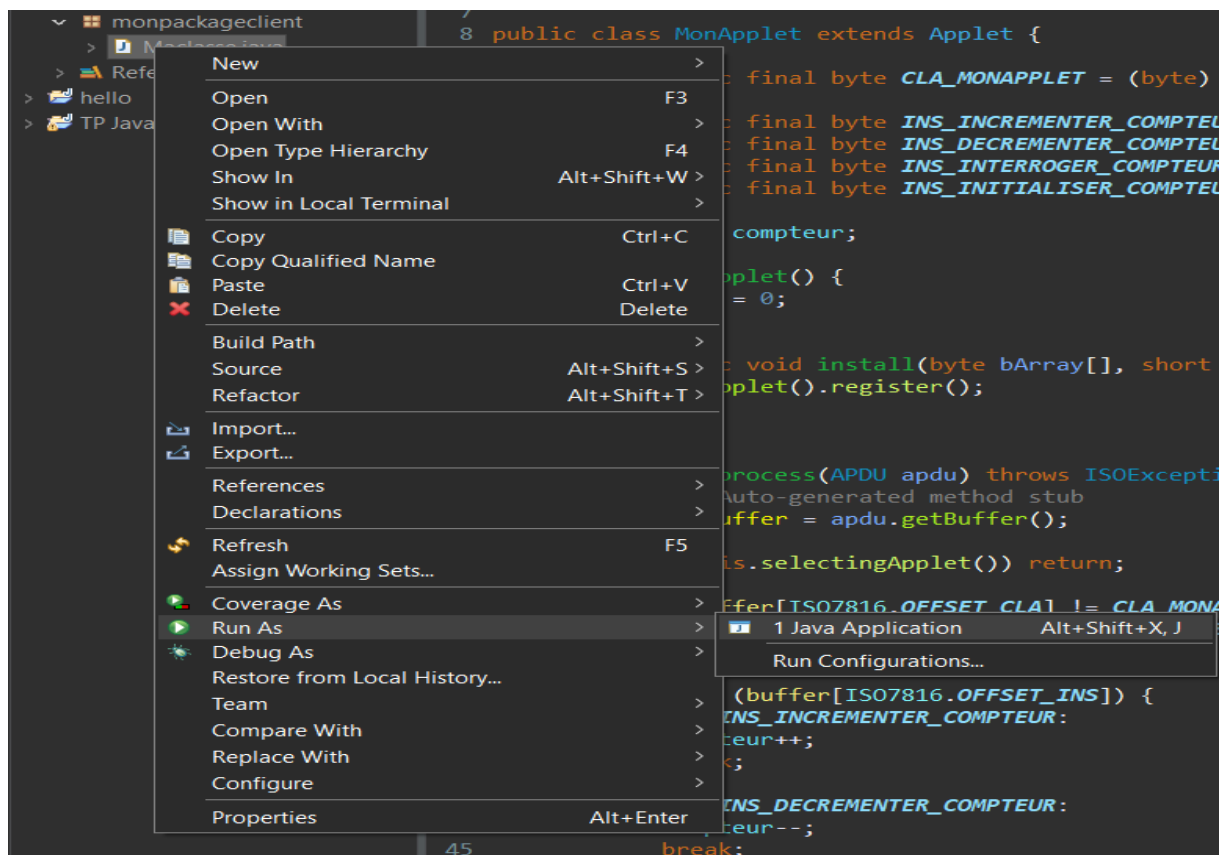
monpackage.MonApplet 0x01:0x02:0x03:0x04:0x05:0x06:0x07:0x08:0x09:0x00:0x00

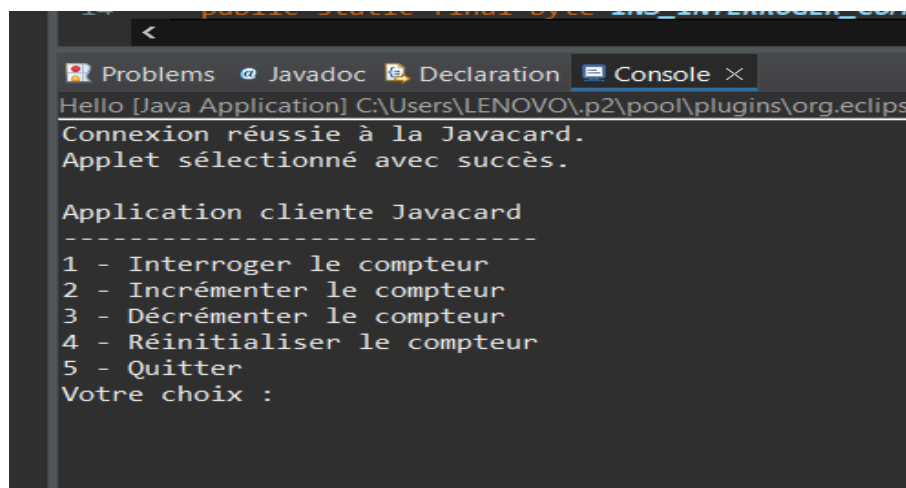
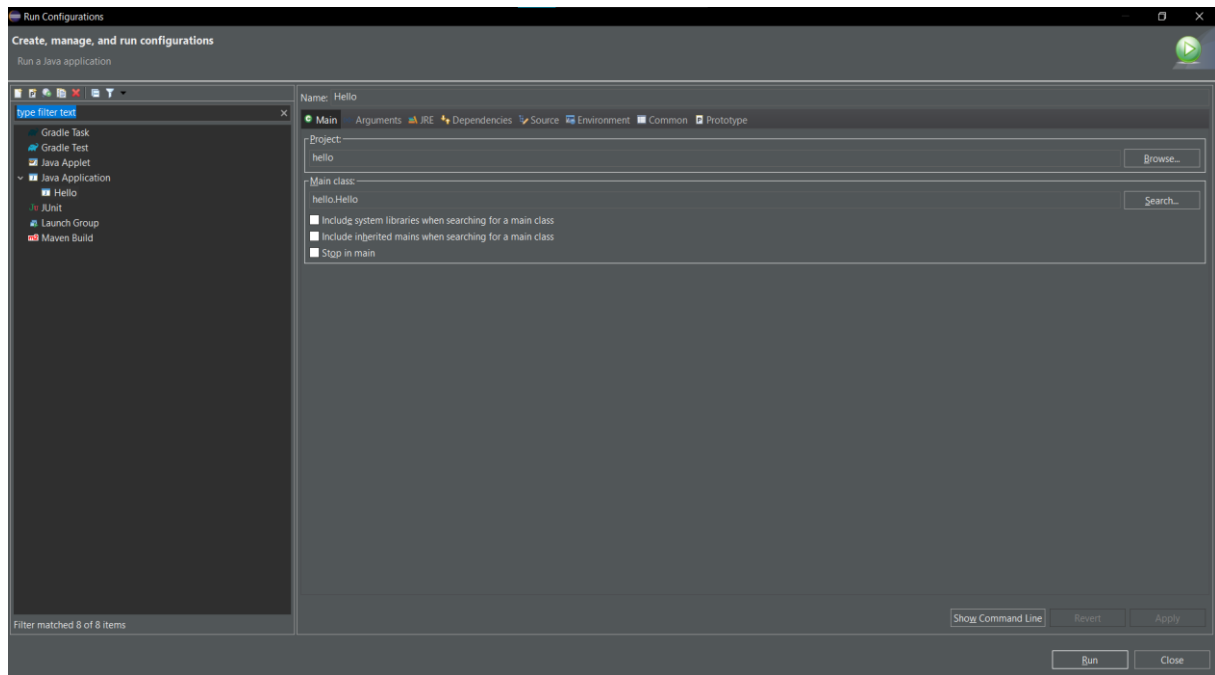


## • Lancer notre simulateur

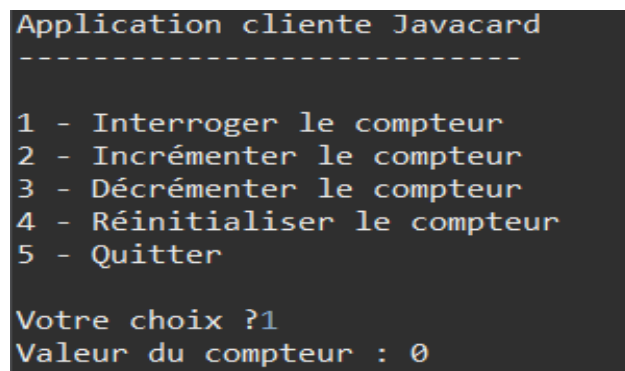


## • Lançons notre application cliente





- **Tester**



```

Votre choix ?2
OK

Application cliente Javacard
-----

1 - Interroger le compteur
2 - Incrémenter le compteur
3 - Décrémenter le compteur
4 - Réinitialiser le compteur
5 - Quitter

Votre choix ?2
OK

Application cliente Javacard
-----

1 - Interroger le compteur
2 - Incrémenter le compteur
3 - Décrémenter le compteur
4 - Réinitialiser le compteur
5 - Quitter

Votre choix ?2
OK

Application cliente Javacard
-----

1 - Interroger le compteur
2 - Incrémenter le compteur
3 - Décrémenter le compteur
4 - Réinitialiser le compteur
5 - Quitter

Votre choix ?1
Valeur du compteur : 3

```

```

Application cliente Javacard
-----

1 - Interroger le compteur
2 - Incrémenter le compteur
3 - Décrémenter le compteur
4 - Réinitialiser le compteur
5 - Quitter

Votre choix ?3
OK

Application cliente Javacard
-----

1 - Interroger le compteur
2 - Incrémenter le compteur
3 - Décrémenter le compteur
4 - Réinitialiser le compteur
5 - Quitter

Votre choix ?1
Valeur du compteur : 2

```

```
Application cliente Javacard
-----

1 - Interroger le compteur
2 - Incrémenter le compteur
3 - Décrémenter le compteur
4 - Réinitialiser le compteur
5 - Quitter

Votre choix ?5
```

```
Copyright 2005 Sun Microsystems, Inc. All Rights Reserved. Use
jcwde is listening for T=1 Adu's on TCP/IP port 9 025.
jcwde exiting on receipt of power down command.

C:\Eclipse\workspace\TP Javacard\bin>_
```

# Mini-Projet

## I. Partie Serveur

### 1. Introduction générale

Implémente un applet Java Card pour un portefeuille électronique sécurisé. Il permet de gérer des opérations telles que le crédit, le débit, la consultation du solde, la vérification et le changement du code PIN. Le portefeuille utilise une clé AES pour sécuriser les transactions et les données sensibles, comme le code PIN. Le code PIN est protégé par un système de tentatives limitées, et les transactions sont soumises à des restrictions de montants et de solde. Chaque opération nécessite la validation du PIN, et des erreurs sont gérées pour garantir la sécurité, comme le contrôle du format du PIN ou des montants de transaction. Le chiffrement AES est utilisé pour déchiffrer le code PIN lors de la vérification et du changement.

### 2. Étape 1. Déclarer les attributs et les constantes

Ce code définit des constantes qui sont utilisées pour la communication avec la carte bancaire via des commandes APDU (Application Protocol Data Units).

- Les constantes de commande sont spécifiées avec des valeurs de type `byte`, représentant les différentes opérations que l'on peut effectuer sur la carte, comme la vérification du PIN (`VERIFY`), le crédit (`CREDIT`), le débit (`DEBIT`), la consultation du solde (`GET_BALANCE`) et le changement de PIN (`CHANGE_PIN`).
- Des constantes supplémentaires définissent des limites pour les transactions, telles que le solde maximal (`MAX_BALANCE`), le montant minimal et maximal des transactions (`MIN_TRANSACTION_AMOUNT`, `MAX_TRANSACTION_AMOUNT`), et les étapes de transaction (`STEP_TRANSACTION_AMOUNT`).
- Le code spécifie également des limites pour le nombre de tentatives de saisie du PIN (`PIN_TRY_LIMIT`) et la taille maximale du PIN (`MAX_PIN_SIZE`), ainsi que plusieurs codes d'état (SW) utilisés pour signaler différents types d'erreurs, comme un échec de la vérification du PIN (`SW_VERIFICATION_FAILED`), une tentative de transaction invalide (`SW_INVALID_TRANSACTION_AMOUNT`), ou une exception de sécurité (`SW_SECURITY_EXCEPTION`). Ces constantes permettent de gérer les échanges avec la carte bancaire de manière structurée et sûre.



### 3. Étape 2. Définition des méthodes publiques qu'elle doit obligatoirement implémenter

Le code de la méthode `process(APDU apdu)` traite les commandes envoyées à l'applet Java Card en fonction des paramètres reçus dans l'APDU (Application Protocol Data Unit). Lorsqu'une commande est reçue, le tableau `buffer` contient les informations pertinentes, comme le code de classe (CLA) et le code d'instruction (INS). Tout d'abord, le code vérifie si la commande est valide en examinant les octets à des positions spécifiques dans le tableau `buffer`. Si le code de classe (CLA) est 0 et le code d'instruction (INS) est `0xA4`, la méthode retourne sans effectuer d'action, indiquant une commande qui n'a pas besoin de traitement supplémentaire. Ensuite, si le code de classe (CLA) ne correspond pas à celui défini pour l'application (`Wallet_CLA`), une exception est lancée, signalant que le type de commande n'est pas pris en charge. Si le code de classe est valide, la méthode vérifie ensuite le code d'instruction (INS) et appelle la méthode correspondante (`getBalance`, `debit`, `credit`, `verify`, `changePin`) pour exécuter l'action demandée. Si l'instruction ne correspond à aucune des options définies, une exception est lancée pour indiquer que l'instruction n'est pas supportée. Cette méthode permet donc de gérer différentes opérations liées au portefeuille, telles que la vérification du solde, les opérations de crédit et de débit, la vérification du PIN, et le changement de PIN.

#### ✓ Méthode `install(byte[] bArray, short bOffset, byte bLength)`

Cette méthode est invoquée lors de l'installation de l'applet sur la carte Java Card. Elle crée une instance de l'applet (`MyApplet`) en utilisant les paramètres `bArray`, `bOffset`, et `bLength`, qui contiennent les données nécessaires au téléchargement de l'applet. Ces paramètres sont typiquement fournis par le système lors du déploiement de l'applet sur la carte.

#### ✓ Méthode `select()`

La méthode `select()` est appelée lorsque l'applet est sélectionnée pour la communication avec un terminal. Elle vérifie si le nombre de tentatives restantes pour entrer le code PIN est supérieur à 0, en utilisant `pin.getTriesRemaining()`. Si des tentatives sont disponibles, la méthode retourne `true`, ce qui permet à l'applet d'être active et prête à traiter les commandes. Si aucune tentative n'est restante, l'applet ne peut pas être sélectionnée.

## ✓ Méthode deselect()

La méthode `deselect()` est appelée lorsque l'applet est désélectionnée, généralement après la fin de la communication avec le terminal. Elle réinitialise l'état du code PIN en appelant `pin.reset()`, ce qui remet à zéro les tentatives restantes pour entrer un code PIN. Cela permet de garantir que l'état du code PIN est propre avant la prochaine utilisation de l'applet.

En somme, ces méthodes gèrent l'installation, la sélection et la désélection de l'applet tout en contrôlant l'état du code PIN pour une sécurité optimale.

## ✓ La Méthode verify(APDU apdu)

La méthode `verify(APDU apdu)` est utilisée pour vérifier le code PIN d'un utilisateur dans une carte Java.

### 1. Réception et Décryptage des Données

La méthode commence par récupérer le buffer des données de la commande via `apdu.getBuffer()` et utilise `apdu.setIncomingAndReceive()` pour recevoir les données entrantes. Elle crée ensuite deux tableaux d'octets temporaires, `tempDecrypted` et `decryptedPin`, en utilisant `JCSysystem.makeTransientByteArray()` pour garantir qu'ils seront effacés lors de la désélection de l'applet.

### 2. Décryptage du PIN

La méthode `decryptData` est ensuite utilisée pour décrypter les données du buffer contenant le PIN crypté à partir de l'offset `ISO7816.OFFSET_CDATA`. Le résultat du décryptage est stocké dans `tempDecrypted`.

### 3. Vérification de la Longueur du PIN

Après le décryptage, le tableau `tempDecrypted` est copié dans `decryptedPin` avec la méthode `Util.arrayCopyNonAtomic()`. Si la longueur du `decryptedPin` ne correspond pas à la taille maximale de PIN (`MAX_PIN_SIZE`), une exception `ISOException` est lancée avec un code d'erreur `SW_INVALID_PIN_LENGTH`.

#### 4. Validation du PIN

Ensuite, la méthode vérifie si le PIN correspond au PIN stocké en appelant `pin.check()`. Si la vérification échoue, une exception est lancée avec un code d'erreur qui inclut le nombre de tentatives restantes pour entrer le PIN (utilisant `pin.getTriesRemaining()`).

#### 5. Gestion des Exceptions

Si une exception quelconque survient lors du processus de décryptage ou de vérification, une exception `ISOException` avec un code d'erreur `SW_SECURITY_EXCEPTION` est lancée pour signaler un problème de sécurité.

## II. Partie Client

### 1. Introduction générale

Ce code implémente un client de carte bancaire en Java qui utilise une interface graphique Swing pour interagir avec l'utilisateur et une carte à puce via un connecteur de communication APDU. Il permet à l'utilisateur de vérifier le PIN de la carte, de créditer ou débiter des fonds, de consulter le solde et de modifier le PIN. La communication avec la carte se fait à l'aide de commandes APDU, qui sont envoyées via un socket à un serveur local. L'application utilise également un chiffrement AES pour sécuriser la saisie du PIN de l'utilisateur. En cas d'erreur de communication ou de validation, des messages d'erreur sont affichés dans l'interface graphique. L'application se termine proprement en fermant la connexion avec la carte à puce avant de quitter.

### 2. Connexion

- **Établissement de la Connexion**

La méthode tente de se connecter à un serveur (applet) sur le port 9025 en créant un Socket via la simulation. Ce socket permet la communication entre l'application et le lecteur de carte à puce via une connexion réseau locale.

- **Création des Flux de Données**

Deux flux sont configurés :

- \* `BufferedInputStream input` pour recevoir les données entrantes.
- \* `BufferedOutputStream output` pour envoyer les données sortantes.

- ✓ **Initialisation du Lecteur de Carte**

Un objet `cad` de type `CadT1Client` est créé en utilisant les flux configurés. Cette classe gère les communications selon le protocole T=1. Ensuite, la méthode `cad.powerUp()` est appelée pour allumer le lecteur de carte.

- ✓ **Gestion des Erreurs**

En cas d'`IOException` (problème de réseau) ou de `CadTransportException` (erreur de communication spécifique au lecteur), un message d'erreur informatif s'affiche. Si l'erreur survient, l'application est immédiatement arrêtée avec `System.exit(1)`.

### 3. Sélection

- ✓ **Préparation de la Commande APDU SELECT**

On prépare la commande de sélection d'applet :

- \* `CLA = 0x00` : Classe de commande standard.
- \* `INS = 0xA4` : Instruction **SELECT** pour sélectionner une application.
- \* `P1 = 0x04` : Sélection basée sur l'AID (Application Identifier).
- \* `P2 = 0x00` : Aucune option supplémentaire.
- \* `setDataIn(APPLET_AID)` : Données de l'AID de l'applet à sélectionner.

- ✓ **Envoi de la Commande**

La commande est envoyée à la carte à puce via `cad.exchangeAdu(selectAdu)`, initiant la communication avec l'applet cible.

- ✓ **Vérification du Statut de Réponse**

Après l'envoi, le statut de la réponse est vérifié :

- \* `0x9000 (Succès)` : L'applet a été sélectionnée avec succès. Un message de confirmation est affiché dans la console.

- \* **Autre statut** : Si le statut est différent, une exception est levée contenant le statut sous forme hexadécimale.

#### ✓ Gestion des Erreurs

En cas d'erreur de communication (`IOException` ou `CadTransportException`), un message d'erreur est affiché dans une boîte de dialogue, et l'application est fermée à l'aide de `System.exit(1)`.

## 4. Invocation des services implémentes

#### ✓ `handleCredit` :

- \* **Vérification du Code PIN**

Le programme commence par vérifier si le code PIN de l'utilisateur est correct. Si cette vérification échoue, l'opération s'arrête immédiatement.

- \* **Saisie du Montant à Créditer**

Une boîte de dialogue demande à l'utilisateur d'entrer un montant à créditer. Ce montant doit être un entier compris entre 10 et 1000 et un multiple de 10.

- \* **Validation du Montant**

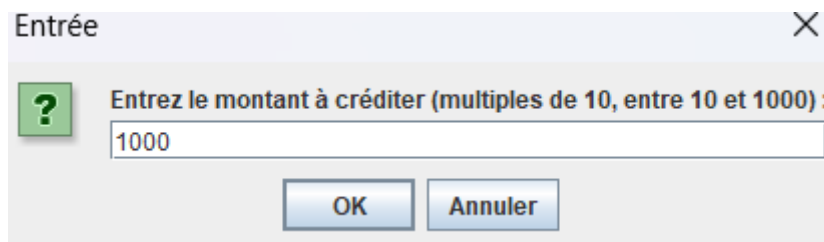
Le montant saisi est converti en entier. Si le montant ne respecte pas les conditions spécifiées, un message d'erreur s'affiche et l'opération s'interrompt.

- \* **Préparation et Envoi de la Commande APDU**

Si le montant est valide, il est converti en deux octets. Une commande APDU contenant ces données est ensuite créée et envoyée à la carte à puce.

- \* **Gestion des Erreurs**

Si l'utilisateur entre un texte non convertible en entier ou un montant invalide, des messages d'erreur appropriés sont affichés pour l'informer du problème.



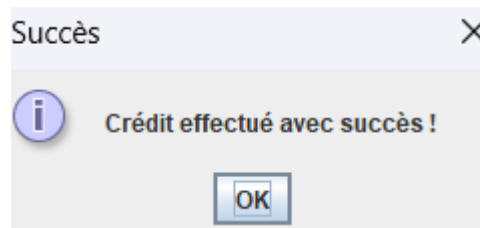
Entrée

?

Entrez le montant à créditer (multiples de 10, entre 10 et 1000):

1000

OK Annuler



## ✓ **handleDebit**

### \* **Vérification du Code PIN**

La méthode commence par vérifier le code PIN de l'utilisateur à l'aide de `verifyPin()`. Si la vérification échoue, l'opération est annulée.

### \* **Affichage des Options de Débit**

Une boîte de dialogue s'ouvre, proposant à l'utilisateur des montants prédéfinis à débiter : 10, 20, 50, 100, 200 ou l'option "Autre". Si l'utilisateur ne fait aucun choix, l'opération est interrompue.

### \* **Saisie et Validation du Montant**

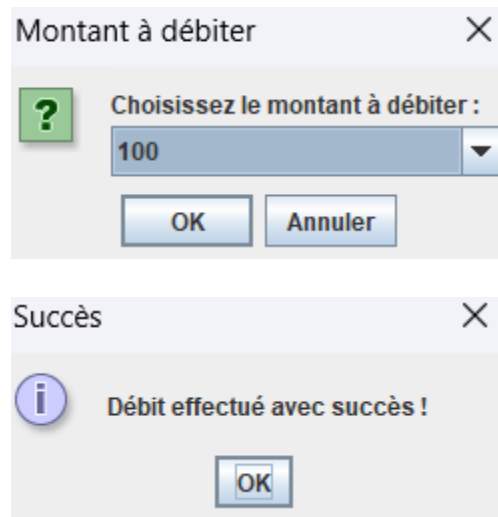
Si l'utilisateur choisit "Autre", il est invité à entrer un montant personnalisé via une boîte de dialogue. Le montant saisi est converti en entier et vérifié : il doit être un multiple de 10, compris entre 10 et 1000. En cas d'erreur de saisie ou de format invalide, un message d'erreur s'affiche, et l'opération est annulée.

### \* **Préparation et Envoi de la Commande APDU**

Si le montant est valide, il est converti en deux octets. Une commande APDU contenant ces données est ensuite créée avec l'instruction `DEBIT` et envoyée à la carte à puce à l'aide de `sendApu(apdu)`.

### \* **Confirmation de l'Opération**

Si l'opération réussit, un message de confirmation indiquant que le débit a été effectué avec succès est affiché à l'utilisateur.



## ✓ **handleGetBalance**

### \* **Vérification du Code PIN**

La méthode commence par vérifier le code PIN de l'utilisateur à l'aide de `verifyPin()`. Si la vérification échoue, l'opération est annulée.

### \* **Création et Envoi de la Commande APDU**

Une commande APDU est créée avec l'instruction `GET_BALANCE` pour demander le solde à la carte à puce. Aucun paramètre supplémentaire n'est nécessaire, donc le champ des données est `null`. La commande est ensuite envoyée via `sendApdu(apdu)`.

### \* **Traitement de la Réponse**

Après l'envoi, le statut de la réponse est vérifié :

- ✓ Si le statut est `0x9000` (succès), les deux octets de données retournés contiennent le solde actuel. Ces octets sont combinés pour calculer le montant total en utilisant un décalage binaire.
- ✓ Si le statut est différent, un message d'erreur s'affiche pour informer l'utilisateur de l'échec de la consultation.

### \* **Affichage du Résultat**

Si l'opération réussit, un message affiche le solde actuel de l'utilisateur dans une boîte de dialogue informative.



✓ **handleChangePin**

\* **Vérification du Code PIN Actuel**

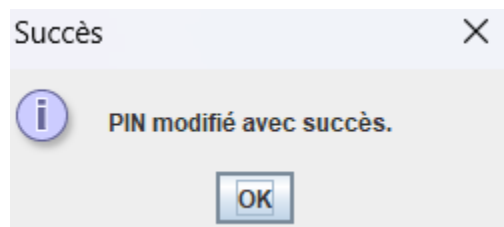
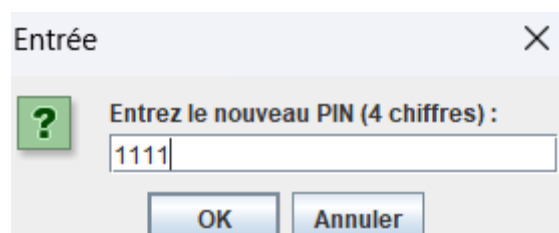
La méthode commence par vérifier le code PIN actuel de l'utilisateur avec `verifyPin()`. Si cette vérification échoue, l'opération est annulée.

\* **Saisie du Nouveau PIN**

Une boîte de dialogue s'affiche pour demander à l'utilisateur de saisir un nouveau code PIN. Le PIN doit contenir exactement **4 chiffres**. Si la saisie est annulée ou invalide (moins de 4 caractères ou caractères non numériques), un message d'erreur est affiché et l'opération s'arrête.

\* **Conversion du PIN en Octets**

Le nouveau PIN est converti en un tableau de 4 octets. Chaque caractère du PIN est converti en sa valeur numérique en soustrayant '0' à son code ASCII. Les valeurs converties sont affichées dans la console pour vérification.





## 5. Mise hors tension

### ✓ Fermeture de l'Application de Carte à Puce

La méthode commence par vérifier si l'objet `cad` (interface de communication avec le lecteur de carte) n'est pas nul. Si c'est le cas, elle appelle la méthode `powerDown()` pour éteindre le lecteur de carte à puce de manière sécurisée.

### ✓ Gestion des Exceptions

Si une exception survient lors de l'extinction du lecteur (par exemple, `IOException` ou `CadTransportException`), un message d'erreur est affiché dans une boîte de dialogue. L'utilisateur est informé de la nature de l'erreur grâce à `e.getMessage()`.

### ✓ Fermeture de l'Application

Après la tentative d'extinction du lecteur, l'application est terminée en appelant `System.exit(0)`, ce qui ferme complètement le programme.

## 6. Create apdu /sendapdu

La méthode `createApu()` est responsable de la création d'une commande APDU (Application Protocol Data Unit) destinée à être envoyée à une carte à puce, tandis que la méthode `sendApu()` se charge de l'envoi de cette commande et de la gestion des erreurs.

### ✓ Création de l'APDU

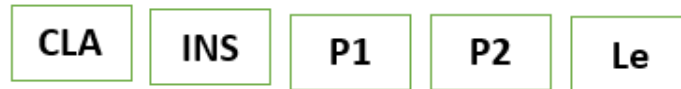
La méthode commence par créer un nouvel objet Apu et initialise les premiers quatre octets de la commande avec les paramètres CLA, INS, P1 et P2, qui représentent respectivement la classe, l'instruction et les paramètres associés à la commande APDU. Ces octets sont ensuite imprimés pour faciliter le débogage. Si des données supplémentaires sont fournies sous forme de tableau `byte[]`, elles sont ajoutées à l'APDU à l'aide de la méthode `setDataIn()`.

Command APDU						
Header (required)				Body (optional)		
CLA	INS	P1	P2	Lc	Data Field	Le

Caso 1:  
No incluye data,  
No requiere respuesta.



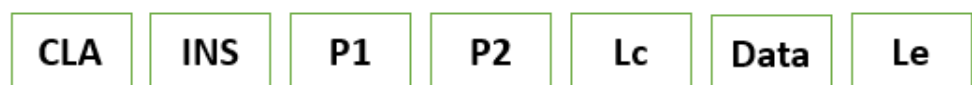
Caso 2:  
No incluye data,  
Requiere respuesta.



Caso 3:  
Incluye data,  
No requiere respuesta.

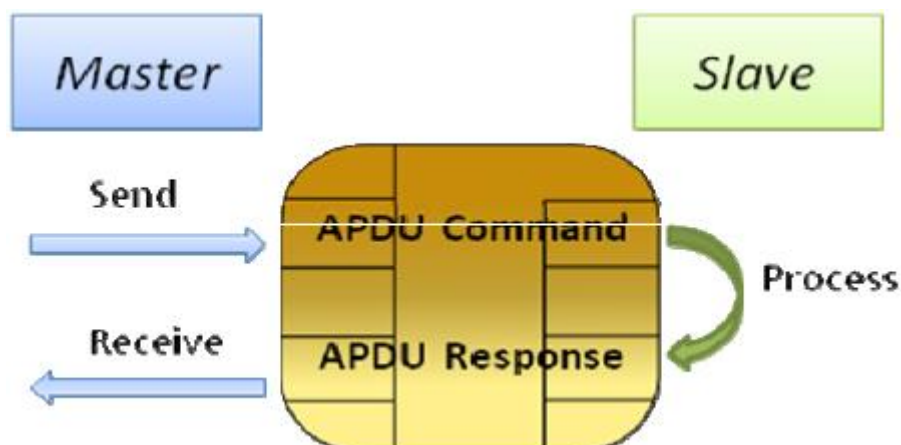


Caso 4:  
Incluye data,  
Requiere respuesta.



## ✓ Envoi de l'APDU

Une fois l'APDU créée, la méthode `sendAdu()` envoie la commande à la carte à puce via l'objet `cad` à l'aide de la méthode `exchangeAdu()`. Si une erreur survient lors de l'envoi de l'APDU (comme une exception liée à la communication ou à l'entrée/sortie), un message d'erreur est affiché à l'utilisateur par le biais d'une boîte de dialogue.



## 7. Verif PIN

La méthode `verifyPin()` est responsable de la vérification du code PIN de l'utilisateur avant d'autoriser toute opération sur la carte à puce. Elle gère plusieurs aspects, dont la saisie du PIN, son chiffrement, l'envoi de la commande APDU et la gestion des tentatives incorrectes.

## ✓ Gestion des tentatives restantes

La méthode commence par vérifier si l'utilisateur a encore des tentatives restantes pour entrer un PIN correct. Si le compteur de tentatives (`pinAttemptsRemaining`) est à zéro, la carte est bloquée, un message d'erreur est affiché, et l'utilisateur ne peut plus continuer. L'état du système est mis à jour pour refléter cette situation ("Carte bloquée").

## ✓ Saisie et validation du PIN

L'utilisateur est ensuite invité à entrer un code PIN à l'aide d'une boîte de dialogue. Si le PIN saisi est vide, n'est pas composé exactement de quatre chiffres ou contient des caractères non numériques, un message d'erreur est affiché, et la méthode retourne `false`, indiquant que la validation a échoué.

## ✓ Chiffrement du PIN

Une fois le PIN validé, chaque caractère du PIN est converti en un tableau d'octets, où chaque chiffre est transformé en son équivalent numérique (par exemple, '3' devient 3). Ce tableau est ensuite chiffré à l'aide de la méthode `encryptPin()`. Si le chiffrement échoue (retourne `null`), la méthode retourne `false`.

## ✓ Envoi de l'APDU de vérification

L'APDU (commande) pour vérifier le PIN est ensuite créée à l'aide de la méthode `createApu()`, en utilisant le tableau `encryptedPin` contenant le PIN chiffré. Cette APDU est envoyée à la carte via la méthode `sendApu()`.


## ✓ Vérification de la réponse

Après l'envoi de l'APDU, la réponse est analysée. Si le statut de la réponse (`apdu.getStatus()`) est `0x9000`, ce qui signifie que la vérification a réussi, un message de succès est affiché, et le compteur de tentatives est réinitialisé à 3. Si la vérification échoue, le nombre de tentatives restantes est décrémenté, et l'utilisateur est informé de l'échec ainsi que du nombre de tentatives restantes. Le statut de la carte est mis à jour pour refléter le PIN incorrect.

Entrée

 Entrez votre PIN (4 chiffres) :


Succès

 PIN vérifié avec succès.

Client Carte Bancaire


**Banque Terminale**

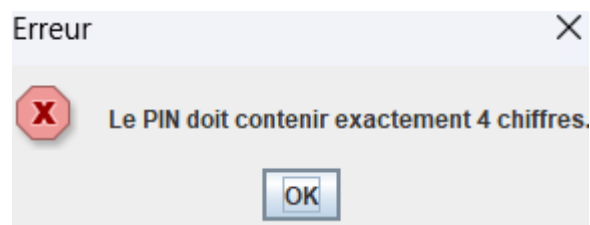
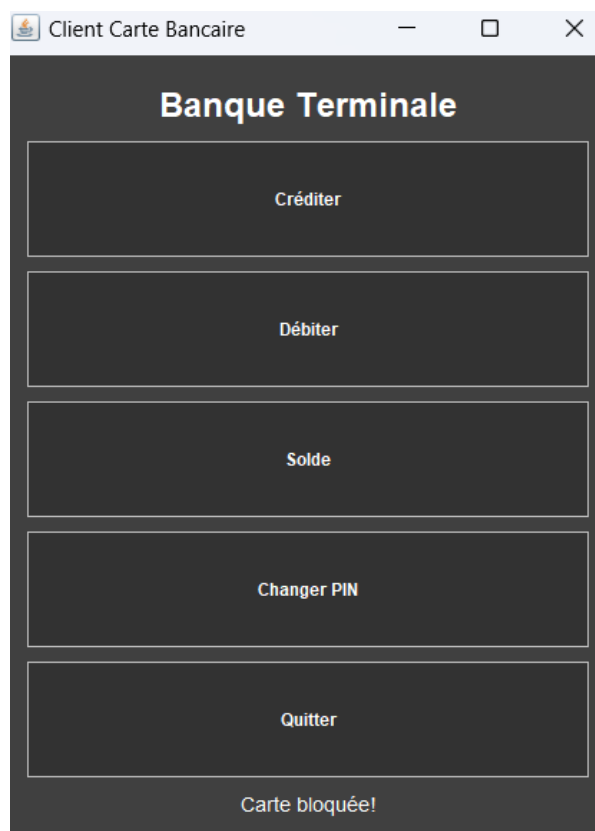
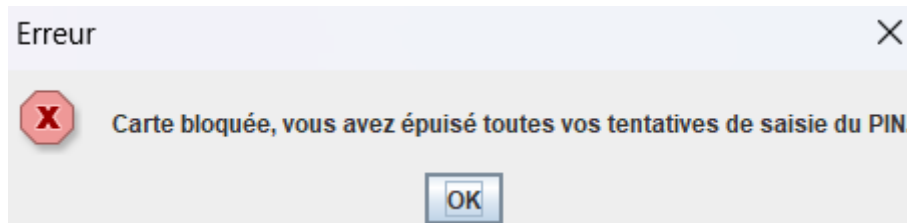
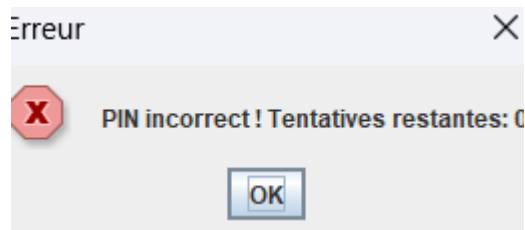
Entrée

 Entrez votre PIN (4 chiffres) :

PIN incorrect ! Tentatives restantes: 2

Erreur

 PIN incorrect ! Tentatives restantes: 1



## 8. Cryptage

La méthode `encryptPin(byte[] pinBytes)` effectue le chiffrement du code PIN à l'aide de l'algorithme AES (Advanced Encryption Standard) en mode CBC (Cipher Block Chaining).

## ✓ Étapes du chiffrement :

### 1. Clé de chiffrement :

- Un tableau d'octets `keyData` est défini pour servir de clé de chiffrement. Cette clé est de 16 octets (128 bits), ce qui est requis par l'algorithme AES pour un chiffrement de taille standard.

### 2. Initialisation du cipher (chiffreur) :

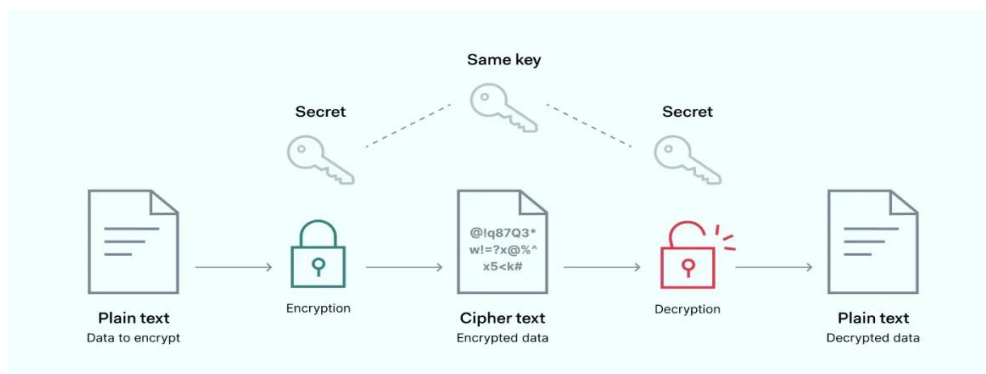
- L'objet `Cipher` est créé avec l'algorithme "AES".
- Notez que cela suppose que le PIN ne dépasse pas 16 octets de toute façon, car AES avec un bloc de 128 bits (16 octets) est utilisé.

### 3. Chiffrement :

- Le chiffreur (`cipher`) est initialisé en mode `ENCRYPT_MODE` avec la clé et l'IV spécifiés. L'opération de chiffrement est effectuée avec `cipher.doFinal(paddedPin)`, qui renvoie le résultat du chiffrement sous forme de tableau d'octets.

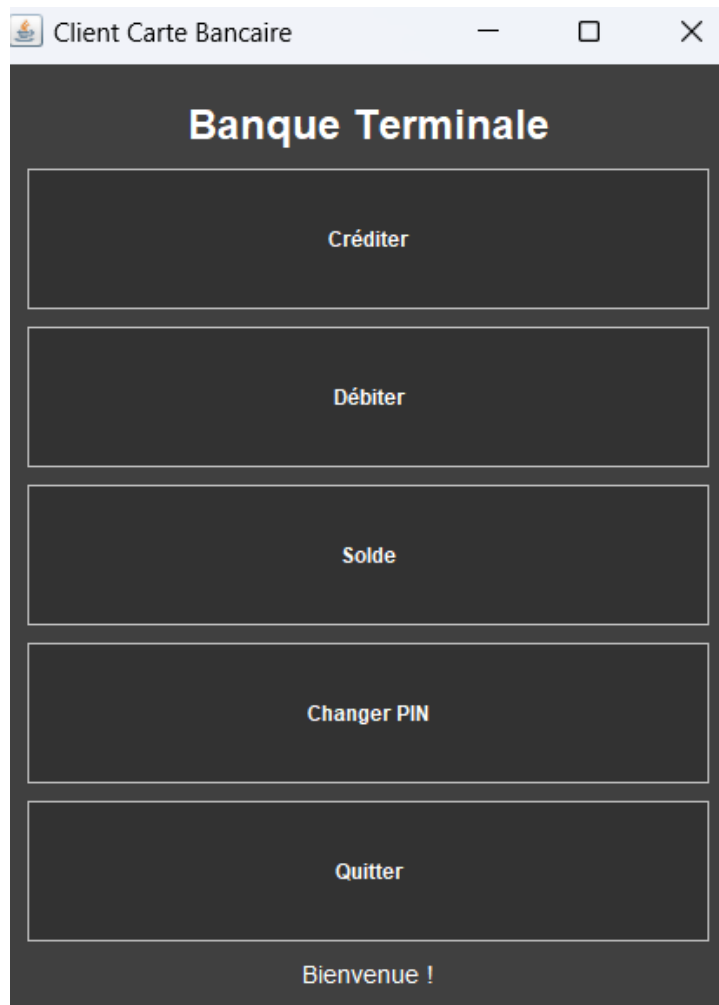
### 4. Gestion des erreurs :

- Si une erreur se produit lors du chiffrement, un message d'erreur est affiché à l'utilisateur via `JOptionPane.showMessageDialog`, et la méthode retourne `null` pour indiquer un échec.



## 9. Préparation de l'interface graphique :

- ✓ Technologie utilisée : Swing
- ✓ Aspect général de l'interface :



Nous créons une application nommée *CardClient*, écrite en Java, qui utilise **Swing** (une bibliothèque graphique) pour concevoir une interface utilisateur et établit une communication avec une carte JavaCard. Au début, nous avons importé différentes bibliothèques nécessaires, comme :

- `javax.swing.*` pour les composants graphiques.

#### ✓ Description de la méthode `createGUI`

La méthode `createGUI` initialise et configure l'interface graphique principale de l'application. Voici les principales étapes de sa mise en œuvre :

- **Création et configuration de la fenêtre principale :**

Une instance de `JFrame` intitulée *Client Carte Bancaire* est créée et configurée avec une taille de 450 x 600 pixels.

La couleur de fond de la fenêtre est définie sur **Color.DARK\_GRAY** pour un style moderne et professionnel.

- **Panneau principal :**

Un panneau principal (JPanel) est créé avec un layout de type BorderLayout.

Ce panneau inclut des bordures internes (à l'aide de `BorderFactory.createEmptyBorder`) pour espacer les éléments de l'interface.

- **Titre :**

Un JLabel est utilisé pour afficher le titre "Banque Terminale".

Le titre est centré (à l'aide de `SwingConstants.CENTER`) et stylisé avec une police **Arial**, en gras, de taille 24.

La couleur du texte est blanche pour contraster avec l'arrière-plan sombre.

- **Panneau des boutons :**

Un sous-panneau (à disposition GridLayout) est ajouté pour contenir les boutons d'action.

Ce panneau est configuré avec 5 lignes et 2 colonnes pour disposer les boutons de manière structurée, avec des espaces de 10 pixels entre eux.

- **Boutons d'action :**

Cinq boutons sont créés : Créditer, Débiter, Solde, Changer PIN et Quitter.

Chaque bouton est stylisé pour ressembler à des boutons de terminaux bancaires :

Couleur de fond : gris foncé (`Color(50, 50, 50)`).

Couleur du texte : blanc.

Police : **Arial**, gras, taille 12.

Bordure : fine, de couleur grise.

Suppression du focus visuel (à l'aide de `setFocusPainted(false)`).

- **Actions des boutons :**

Des gestionnaires d'événements (ActionListener) sont associés à chaque bouton pour appeler les méthodes correspondantes :

`handleCredit` : pour créditer un montant.

`handleDebit` : pour débiter un montant.

`handleGetBalance` : pour consulter le solde.

`handleChangePin` : pour changer le PIN.

`closeApplication` : pour quitter l'application.

- **Label de statut :**



Un JLabel est ajouté en bas de la fenêtre pour afficher des messages de statut.

Par défaut, il affiche "Bienvenue !" avec une police **Arial** de taille 14 et une couleur blanche.

- **Affichage final :**

Le panneau principal est ajouté à la fenêtre (JFrame).

La fenêtre est centrée sur l'écran (à l'aide de setLocationRelativeTo(null)) et rendue visible.