

2024 - 2025



JavaCard :

Programmation d'un

Distributeur

Automatique Sécurisé

"supervisé par le professeur : Ramzi Mahmoudi"



Réalisé par : Sirin GRIRA & Syrine AMMAR & Roua BEN TIBA
ING1 INFO

Bilan:

*Réalisé par : Sirin GRIRA & Syrine AMMAR & Roua BEN TIBA
ING1 INFO*

1

Guide de Telechargement et de Configuration de JavaCard

2

Développement d'une Application Côté Serveur - V2.0

3

Développement d'une Application Côté Client - V2.0

4

Mini projet : Distributeur JavaCard

TP1 -Environnement de développement JavaCard

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.1 Preparation des outils logiciels nécessaires :

 eclipse-inst-jre-win64.exe	16/09/2024 1:51 PM	Application	135 499 Ko
 java_card_kit-2_2_2-windows.zip	16/09/2024 1:33 PM	Dossier compressé	29 606 Ko
 eclipse-jcde-0.2.zip	16/09/2024 11:02 AM	Dossier compressé	144 Ko
 jre-7u80-windows-x64.exe	16/09/2024 11:01 AM	Application	30 508 Ko
 jdk-7u80-windows-x64.exe	16/09/2024 11:00 AM	Application	143 451 Ko

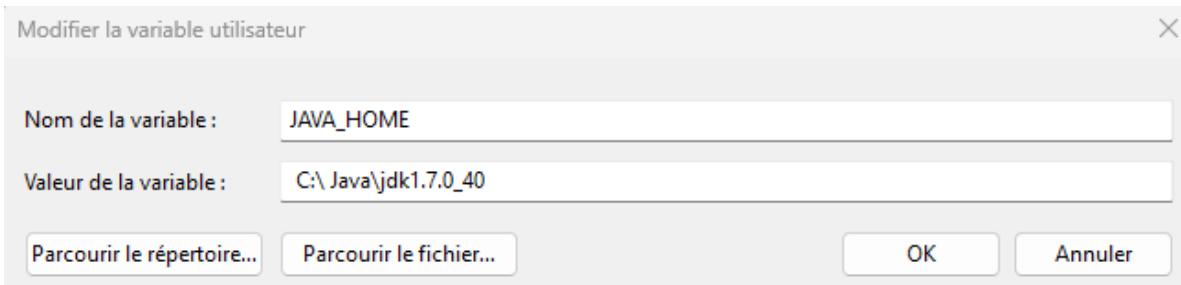
I.2 Instructions d'installation :

I.2.1 Installation d'Eclipse sous windows :

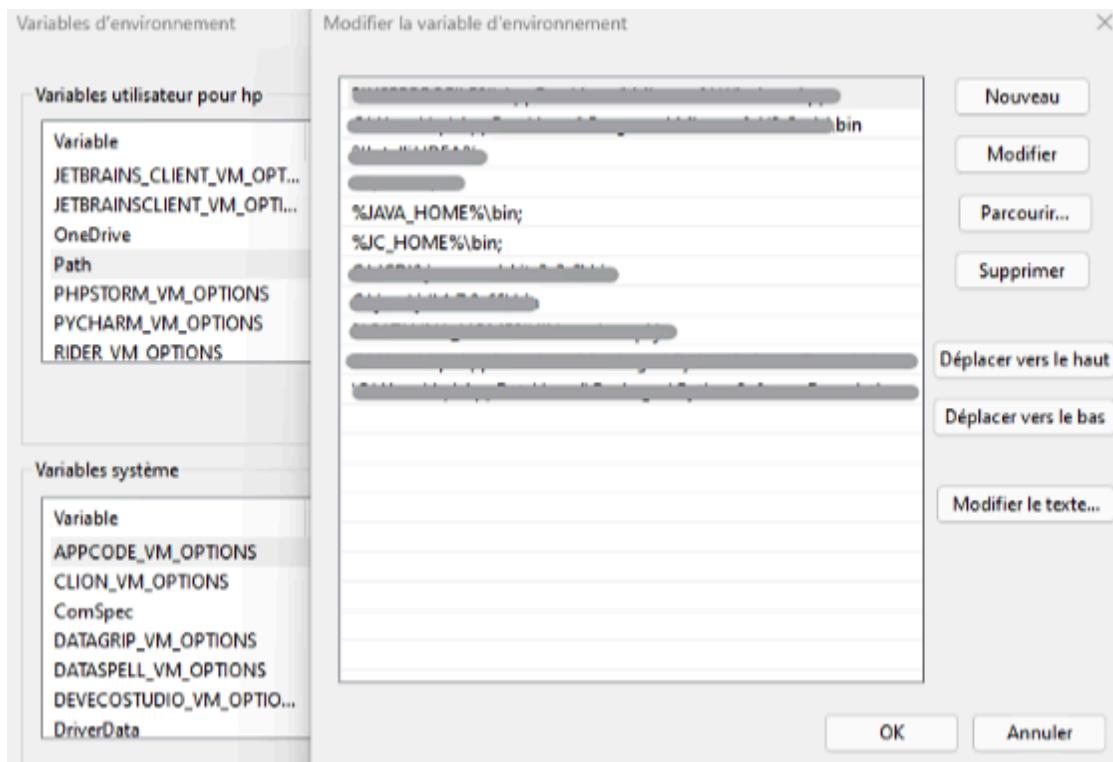
a. Installation du JDK

 eclipse-inst-jre-win64.exe	16/09/2024 1:51 PM	Application	135 499 Ko
 java_card_kit-2_2_2-windows.zip	16/09/2024 1:33 PM	Dossier compressé	29 606 Ko
 eclipse-jcde-0.2.zip	16/09/2024 11:02 AM	Dossier compressé	144 Ko
 jre-7u80-windows-x64.exe	16/09/2024 11:01 AM	Application	30 508 Ko
 jdk-7u80-windows-x64.exe	16/09/2024 11:00 AM	Application	143 451 Ko

3. Créer une nouvelle variable d'environnement (coté utilisateur) nommée JAVA_HOME contenant le chemin C:\ Java\jdk1.7.0_40



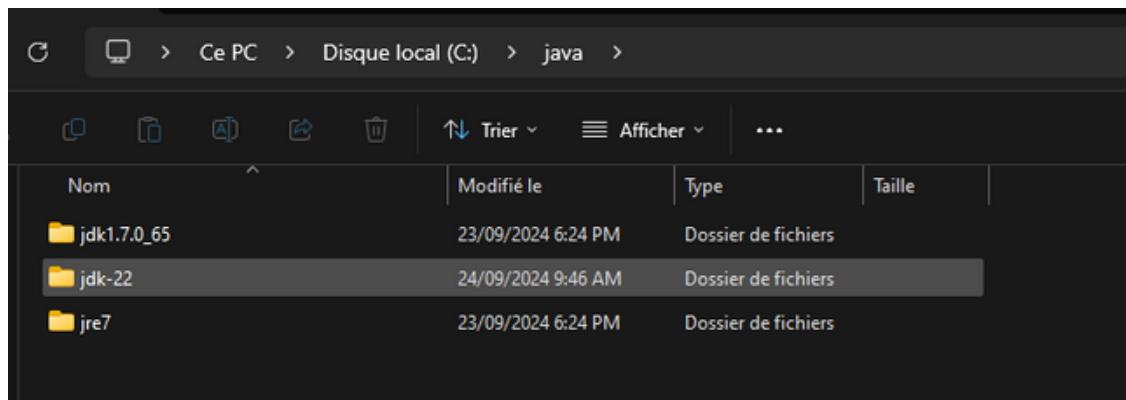
4. Modifier la variable Path (coté système) en ajoutant vers sa fin :
%JAVA_HOME%\bin;



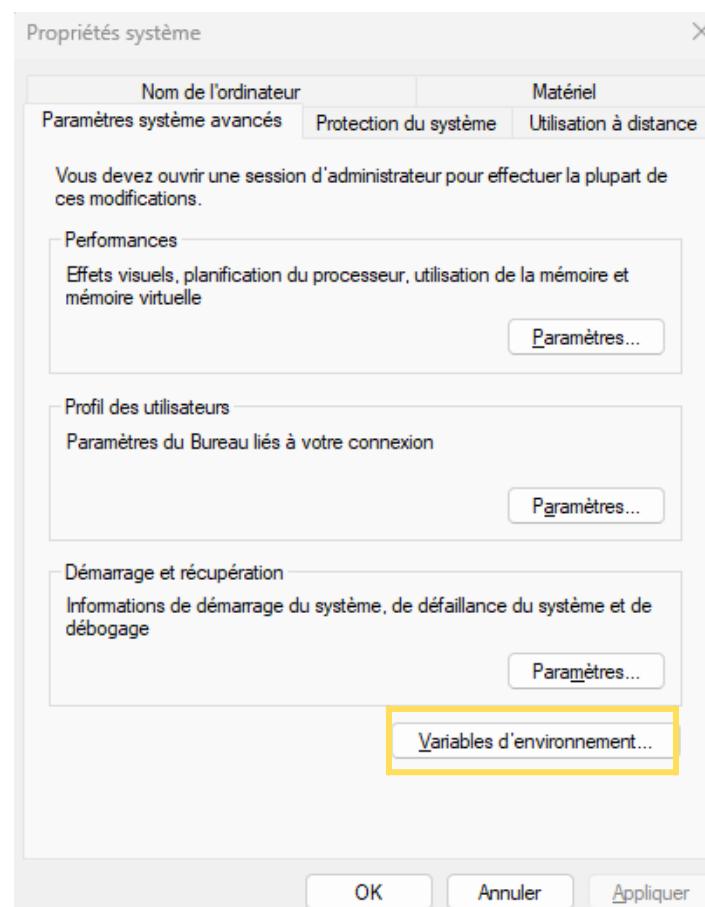
5. Pour vérifier le bon fonctionnement de votre JDK, il suffit de taper la commande « java –version » sous la console DOS :

```
>java --version
java 22.0.2 2024-07-16
Java(TM) SE Runtime Environment (build 22.0.2+9-70)
Java HotSpot(TM) 64-Bit Server VM (build 22.0.2+9-70, mixed mode, sharing)
```

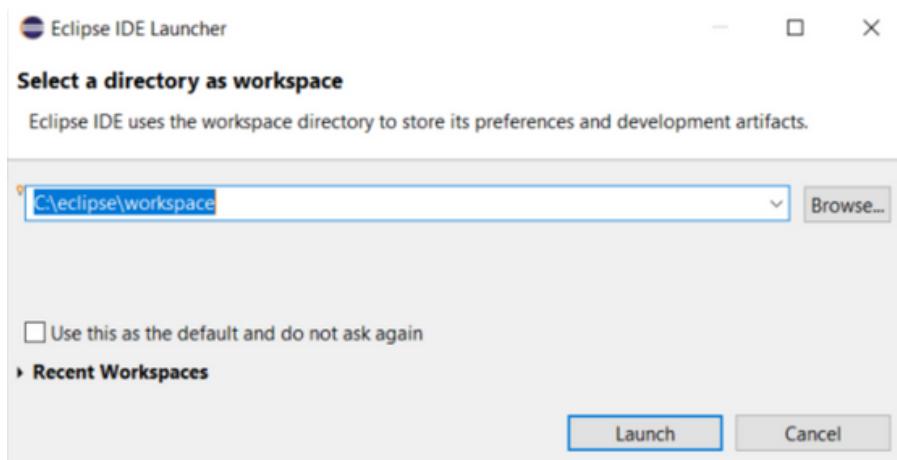
1. Créer un nouveau répertoire C:\Java, puis recopier tout le contenu du répertoire C:\Program Files\Java\ sous ce répertoire C:\Java.



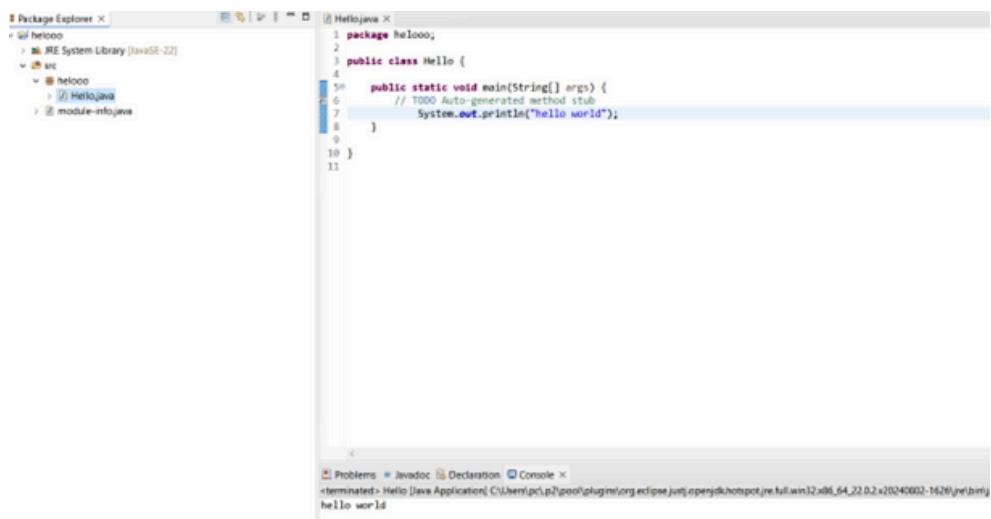
2. Cliquer avec le bouton droit sur l'icône du Poste de travail, puis Propriétés > Paramètres système Avancés > Variables d'environnement



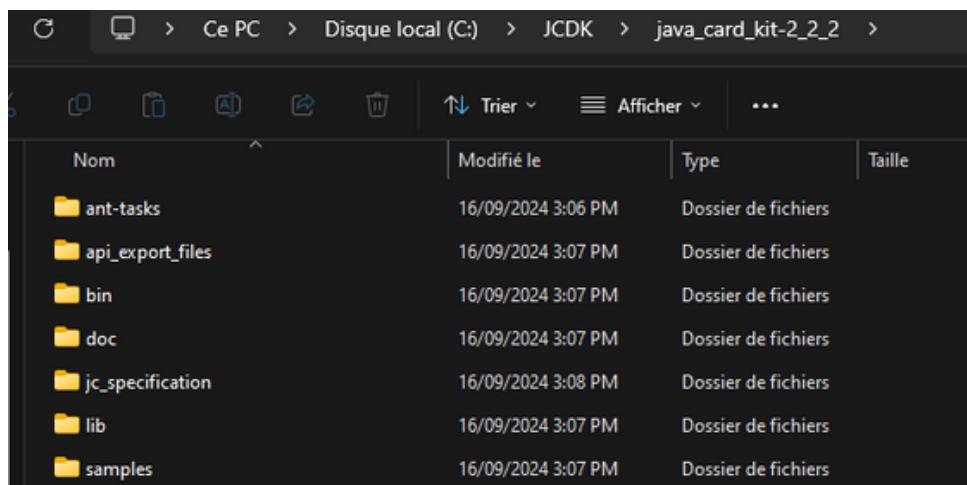
b. Installation de l'IDE – Eclipse :



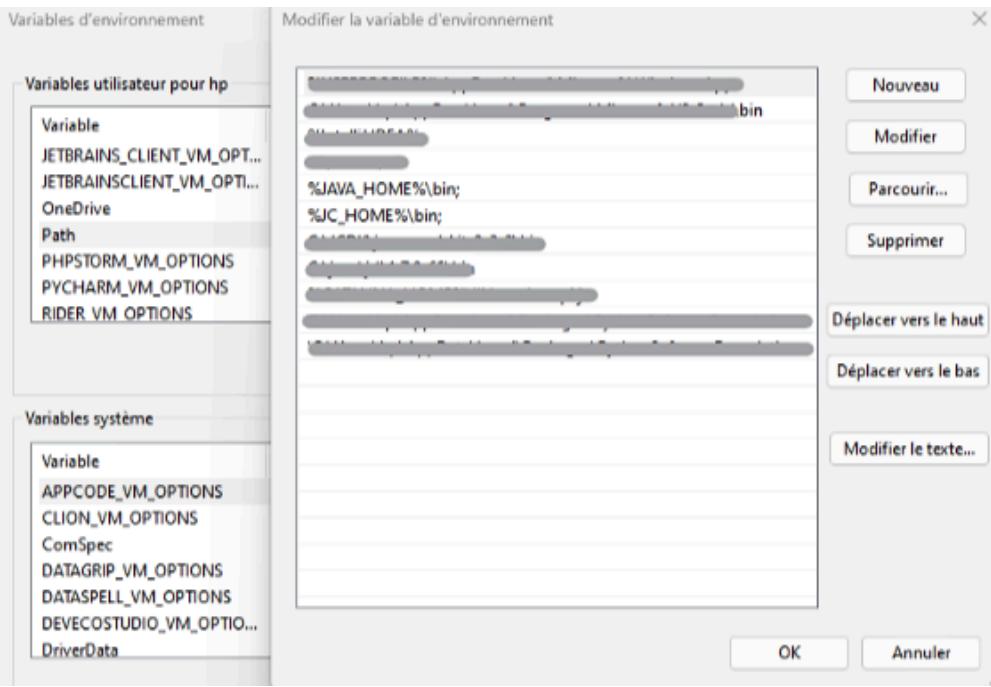
c. L'indispensable « Hello Word »



I.2.2 Installation du Java Card Development Kit 2.2.2 :



Une fois l'installation terminée, il faut ajouter la variable d'environnement JC_HOME contenant le chemin d'accès au JCDK dans la définition de la variable Path.



Vérifiez la réussite de configuration en tapant « apdutool » dans une console de commandes :

```
>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
```

I.2.3 Mise à jour des plugins Eclipse-JCDE :

L'installation consiste à décompresser l'archive eclipse-jcde-0.2.zip après l'avoir téléchargée dans le répertoire C:\Eclipse\eclipse\plugins. Un sous-dossier "plugins" sera créé, il faut simplement recopier tout son contenu directement sous C:\Eclipse\eclipse\plugins.

Nom	Modifié le	Type	Taille
.DS_Store	31/07/2012 10:31 AM	Fichier DS_STORE	7 Ko
org.eclipsejcde.core_0.2.0.jar	31/07/2012 10:26 AM	Executable Jar File	48 Ko
org.eclipsejcde.cref_0.1.0.jar	21/09/2006 11:38 PM	Executable Jar File	9 Ko
org.eclipsejcde.jcbp.wizards_0.1.0.jar	21/09/2006 11:38 PM	Executable Jar File	9 Ko
org.eclipsejcde.jcbp_0.1.0.jar	21/09/2006 11:38 PM	Executable Jar File	22 Ko
org.eclipsejcde.jctools_0.1.0.jar	21/09/2006 11:38 PM	Executable Jar File	21 Ko
org.eclipsejcde.jcwde_0.1.0.jar	21/09/2006 11:38 PM	Executable Jar File	10 Ko
org.eclipsejcde.preferences_0.1.0.jar	21/09/2006 11:38 PM	Executable Jar File	8 Ko
org.eclipsejcde.wizards_0.1.0.jar	21/09/2006 11:38 PM	Executable Jar File	34 Ko

TP2 – Développement d'une application coté serveur – V 2.0

But de ce TP :

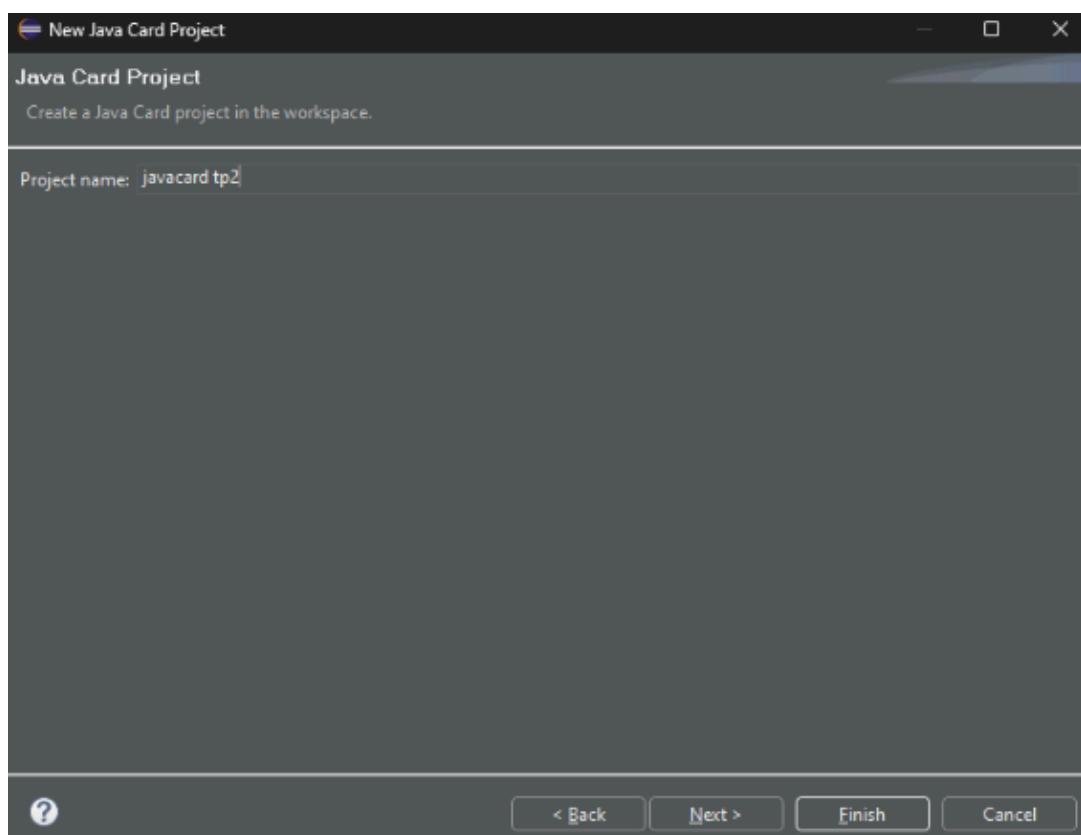
Ce TP a pour but de développer une première applet Java Card (appelée parfois cardlet car elle s'exécute sur la carte à puce), et de les tester à l'aide d'un simulateur de carte.

I. Les différentes étapes de développement

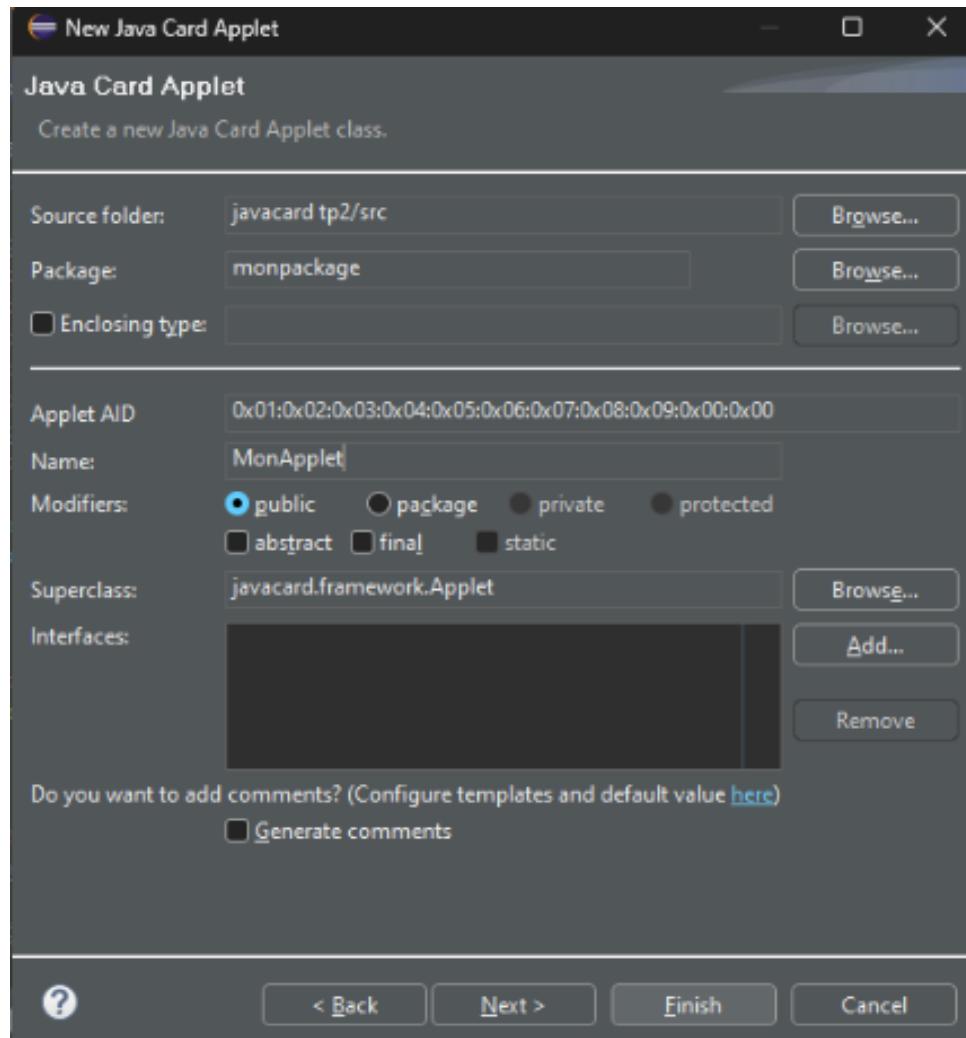
I.1 Programmation de l'application Serveur

I.1.1 Création de l'applet card sous Eclipse :

1. Crédit d'un nouveau projet :



2. Création d'une applet Javacard:



I.1.2 Codage de notre applet :

Étape 1. Ajouter API JavaCard :

```
*MonApplet.java ×
1 package monpackage;
2
3 import javacard.framework.APDU;
4 import javacard.framework.Applet;
5 import javacard.framework.ISO7816;
6 import javacard.framework.ISOException;
```

Étape 2. Déclarer les attributs et les constantes :

```
public class MonApplet extends Applet {

    /* Constantes */
    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;
    /* Attributs */
    private byte compteur;
```

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

1. la méthode **install ()** : création et enregistrement de l'objet Applet
2. la méthode **process ()** : Traitement des commandes APDU

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

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

@Override
public void process(APDU apdu) throws ISOException {
    //Extraire le buffer APDU
    byte[] buffer = apdu.getBuffer();
    //Vérifier les octets entête d' APDU
    if (this.selectingApplet()) {
        return;
    }
    if (buffer[ISO7816.OFFSET_CLA] != CLA_MONAPPLET) {
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
    }
    //Traiter la commande APDU
    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;
            //Renvoyer le stat d'état (word status)
    default:
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }
}
```

I.1.3 Outils de simulation :

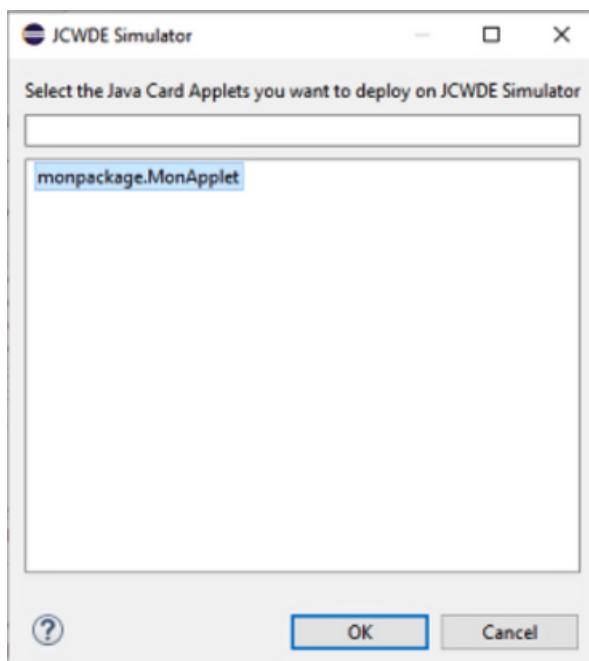
a) APDUTOOL : envoi/réception d'APDU

L'outil **APDUTOOL** du JCDK permet d'échanger des APDU avec une Javacard réelle ou un simulateur. Nous l'utiliserons dans la suite de ce TP.

b) JCWDE : simulateur sans conservation d'état

Le simulateur JCWDE ne nécessite pas de phase d'upload de code. Il nous suffit juste d'installer puis de sélectionner notre applet pour pouvoir la faire fonctionner.

- Dans le menu **JCWDE** d'Eclipse, sélectionner **Start** :



- Cliquer sur le bouton OK pour lancer le simulateur JCWDE avec notre applet. Le simulateur est alors lancé et attend une connexion. Nous pouvons alors lancer l'outil **APDUTOOL**. Pour cela, ouvrir une invite de commandes, lancer le script setvars.bat, puis taper **apdutool** et valider :

```
Administrator: >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.
```

- Installons notre applet en recopiant les **APDU** contenus dans le script **create-MonApplet.script** (sauf la dernière instruction **powerdown**) :

```
Administrator : Invite de commandes - apdutool
Connected.
powerup;
// Select the installer applet
0x00 0xA4 0x04 0x00 0x09 0xa0 0x00 0x00 0x00 0x62 0x03 0x01 0x08 0x01 0x7F;
// create MonApplet applet
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00
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 0x
0 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:
90
```

- Nous pouvons alors sélectionner notre applet en recopiant l'**APDU** contenu dans **select-MonApplet.script** :

```
powerup;
// select MonApplet applet
0x00 0xA4 0x04 0x00 0xb 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x00 0
x7F;
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00
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, SW2: 00
```

- Notre applet est désormais sélectionnée et nous pouvons la tester. Commençons par **interroger le compteur (INS = 0x02)** en envoyant l'**APDU** suivante :

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

Nous pouvons voir que le compteur est pour l'instant à 0

- Nous allons incrémenter le compteur (INS = 0x00) en envoyant l'APDU suivante :

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

- Interrogeons de nouveau le compteur (INS = 0x02) en envoyant l'APDU suivante :

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

Le compteur est désormais à 1.

- Initialisons maintenant (INS = 0x03) le compteur à **0x4A** en envoyant l'APDU suivante :

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

- Décrémentons le compteur (INS = 0x01) en envoyant l'APDU suivante :

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

- Interrogeons de nouveau le compteur (INS = 0x02) en envoyant l'APDU suivante :

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

0x4A - 1 = 0x49. Le résultat est cohérent.

- Ensuite, pour terminer votre session avec le simulateur JCWDE, vous avez exécuté la commande "powerdown;" dans APDUTOO

powerdown;

c) CREF : simulateur avec conservation d'état

JCWDE a l'avantage de ne pas nécessiter l'upload de l'applet sur le simulateur, ce qui le rend plus souple par rapport à **CREF**. En revanche dès que **JCWDE** est stoppé (à la réception d'une commande powerdown par exemple), l'état de la carte n'est pas conservé (exemple : la valeur du compteur dans le cas de l'applet de ce TP).

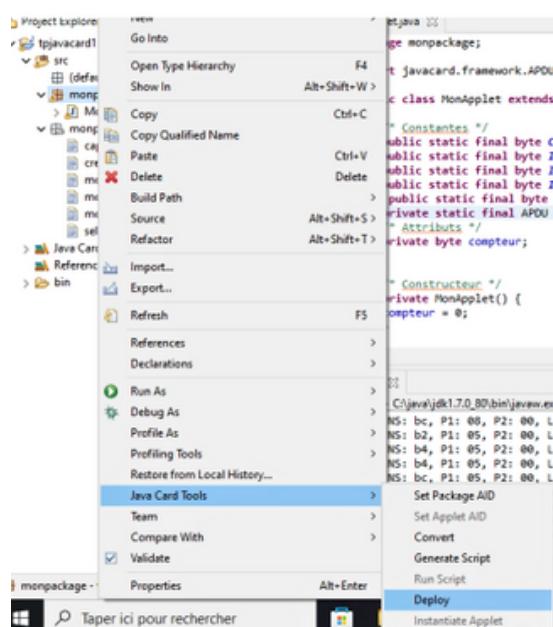
- Nous pouvons lancer **CREF** et créer notre fichier image. Pour cela, ouvrir une invite de commandes, taper "**cref -o monapplet.eeprom**" et valider

```
>>> 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   0xffff
  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
```

- Uploadons maintenant notre applet. Pour cela, clic droit sur le package **monpackage**, Java Card Tools, Deploy :



- Nous pouvons voir les APDUs envoyés dans la zone Console d'Eclipse

```
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x01
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 09, a0, 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, 01, 02, 03, 04, 05, 06, 07, 08, 09, 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,
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, 05, 15, 02, 03, 01, 07, a0, 00, 00, 62, 01, 01, 00, 01, 07, a0, 00, 00, 00, 62, 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: 05, 06, 00, 00, 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, 10, 03, 08, 00, 7a, 02, 30, 8f, 00, 02, 3d, 8c, 00,
CLA: 80, INS: bc, 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,
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,
CLA: 80, INS: bc, P1: 07, P2: 00, Lc: 20, 03, se, 00, 38, 19, 03, 04, 8b, 00, 08, 70, 15, 19, 8b, 00, 09, 3b, 1a, 08, 25, 88, 00,
```

- Relançons CREF afin d'installer notre applet, en prenant soin de recharger notre fichier image (option -i)

```
c:\>cref -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   0xffff
  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 (0xffe0 bytes) restored from file "monapplet.eeprom"
Using a pre-initialized Mask
```

- Installons notre applet. Pour cela, dans Eclipse, clic droit sur le script `create-MonApplet.script`, Java Card Tools, Run Script :

```
<terminated> C:\Java\jdk1.7.0_65\bin\javaw.exe [21 oct. 2014 12:38:14 AM] [pid: 6500]
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, 62, 03, 01, 08, 01, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b6, P1: 00, P2: 00, Lc: 0d, 0b, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, 00, Le: 0b, 01, 02, 03, 04, 05, 06, 07, 08,
```

- À ce stade, nous pouvons utiliser apdutool pour tester notre carte, de la même manière qu'avec JCWDE, en prenant soin de ne pas oublier la commande powerup et la sélection de l'applet. Commençons par relancer CREF (qui s'est terminé sur une commande powerdown)

```

>cref -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    0xffff
  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 (0xffe0 bytes) restored from file "monapplet.eeprom"
using a pre-initialized Mask

```

- Installons notre applet. Pour cela, dans Eclipse, clic droit sur le script create-MonApplet.script, Java Card Tools, Run Script :

```

Administrator : Invite de commandes - apdutool
Microsoft Windows [version 10.0.22631.4317]
(c) Microsoft Corporation. Tous droits réservés.

C:\Windows\System32>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;
// Select the installer applet
0x00 0xA4 0x04 0x00 0x09 0xa0 0x00 0x00 0x62 0x03 0x01 0x08 0x01 0x7F;
// create MonApplet applet
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
0x80 0xB8 0x00 0x00 0xd 0xb 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x00 0x00 0x7F;
CLA: 00, INS: b8, P1: 00, P2: 00, Lc: 0d, 0b, 01, 02, 03, 04, 05, 06, 07, 08, 09, 00, 00, 00, Le: 00, SW1: 64, SW2: 44

```

TP3 - Développement d'une Application Côté Client - V2.0

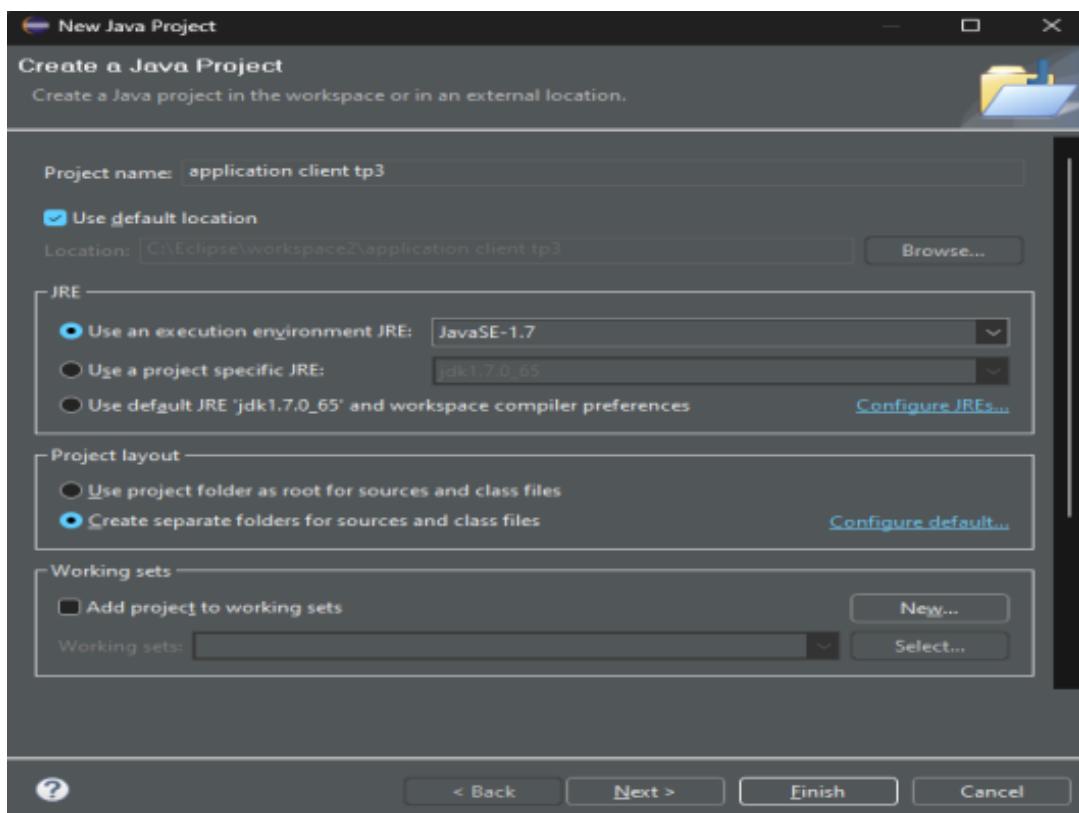
But de ce TP :

Maintenant que nous avons programmé notre applet Javcard compteur, nous pouvons coder une application cliente : l'équivalent du terminal bancaire si notre Javacard était une carte de paiement.

I.1 Crédation de l'application client sous Eclipse

a) Création d'un nouveau projet :

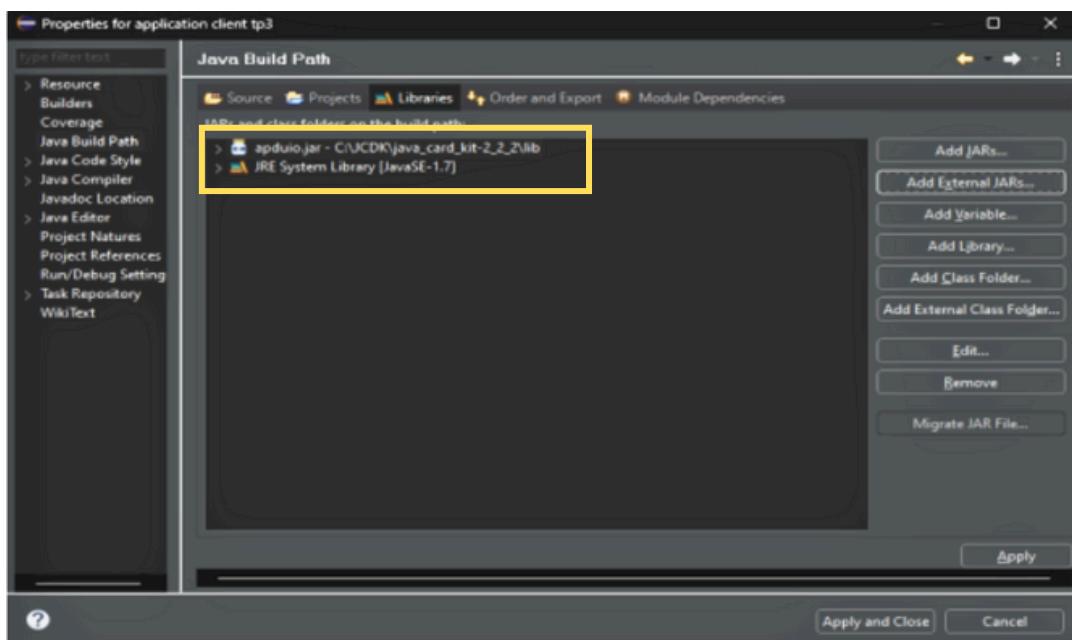
- **Créons un nouveau projet. Pour cela, dans Eclipse, aller dans le menu File, faire New puis Project... :**



b) Ajout de la librairie « apduio » dans le classpath

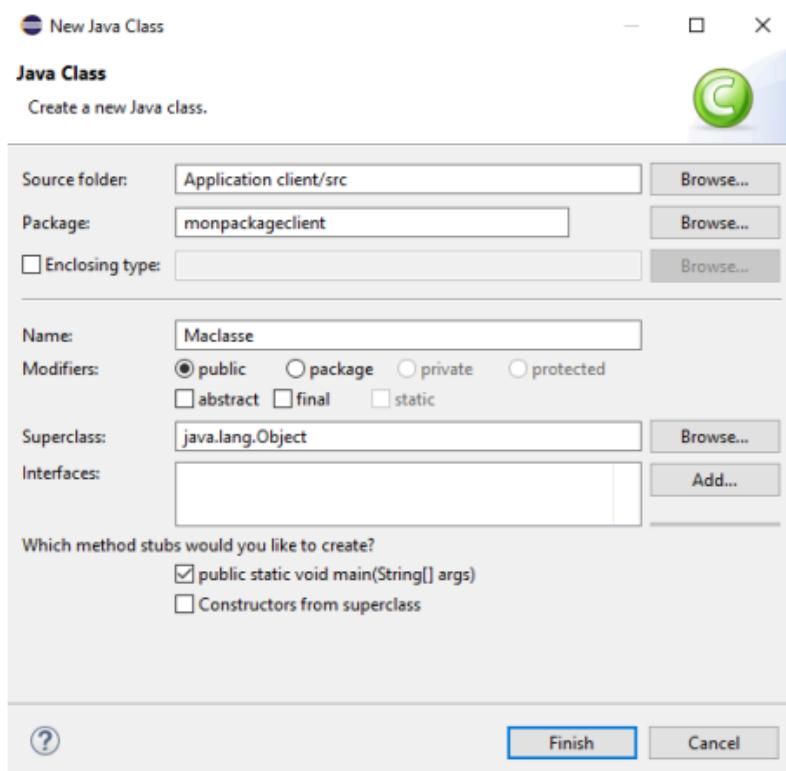
Afin de pouvoir utiliser les classes servant à communiquer avec notre Javacard, il faut ajouter la bibliothèque **apduio.jar** (présente dans le répertoire C:\JCDK\java_card_kit-2_2_2\lib).

- Pour cela, faire un clic droit sur notre projet **Application Client** puis **Propriétés...**
- Dans la section **Java Build Path**, sélectionner l'onglet **Librairies** et cliquer sur le bouton **Add External Jars ...**
- Sélectionner alors le fichier **apduio.jar**, valider et appuyer sur le bouton **OK**.



c) Création de la classe principale :

- Tout d'abord, créons un nouveau package. Pour cela, faire un clic droit sur notre projet, puis **New et Package**
- Donner un nom au package (**monpackageclient** par exemple), puis valider à l'aide du bouton **Finish**
- Créons maintenant la classe principale de notre application. Pour cela, faire un clic droit sur le package créé puis **New et Class**
- Donnons un nom à notre nouvelle classe (**Maclasse** par exemple), cocher la case **public static void main** puis cliquer sur **Finish**



L'application cliente se trouve sur le terminal qui communique avec le serveur (applet carte), on peut séparer l'écriture de notre application en plusieurs étapes :

Étape 1 - Connexion

La connexion au simulateur se fait via une socket. Le simulateur écoute par défaut sur le port 9025. La classe que nous utiliserons pour les échanges de données est CadT1Client.

```
package monpackageclient;

import java.io.BufferedReader;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.IOException;
import java.net.Socket;

import com.sun.javacard.apduio.Apdu;
import com.sun.javacard.apduio.CadT1Client;
import com.sun.javacard.apduio.CadTransportException;

public class Maclasse {

    public static void main(String[] args) {
        /* Connexion - Javacard */
        CadT1Client cad;
        Socket sckCarte;
        try {
            sckCarte = new Socket("localhost", 9025);
            sckCarte.setTcpNoDelay(true);
            BufferedReader input = new BufferedReader(new InputStreamReader(sckCarte.getInputStream()));
            BufferedWriter output = new BufferedWriter(new OutputStreamWriter(sckCarte.getOutputStream()));
            cad = new CadT1Client(input, output);
        } catch (Exception e) {
            System.out.println("Erreur : impossible de se connecter à la Javacard");
            return;
        }
        /* Mise sous tension de la carte */
        try {
            cad.powerUp();
        } catch (Exception e) {System.out.println("Erreur lors de l'envoi de la commande Powerup à la Javacard");
            return;
        }
    }
}
```

- **Socket** utilisée pour simuler une connexion en local sur le port 9025. C'est comme si vous branchiez physiquement une carte à un terminal bancaire
- **CadT1Client** la classe responsable aux échanges entre le client (ordinateur) et la carte via des commandes spécifiques
- **Cad.powerUp()** nécessaire avant toute interaction pour activer la carte virtuelle

Etape 2 - Sélection

La sélection d'applet se fait en envoyant la commade SELECT APDU (utilisée pour sélectionner un applet via son AID (Application Identifier)

- On va choisir l'applet qui sera utilisée pour les opérations

```

/* Sélection de l'applet */
Apdu apdu = new Apdu();
apdu.command[Apdu.CLA] = 0x00;
apdu.command[Apdu.INS] = (byte) 0xA4;
apdu.command[Apdu.P1] = 0x04;
apdu.command[Apdu.P2] = 0x00;
byte[] appletAID = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
0x07, 0x08, 0x09, 0x00, 0x00 };
apdu.setDataIn(appletAID);
try {
    cad.exchangeApdu(apdu);
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (CadTransportException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
if (apdu.getStatus() != 0x9000) {
    System.out.println("Erreur lors de la sélection de l'applet");
    System.exit(1);
}

```

Etape 3 - Invocation des services implémentés

Il suffit pour chaque opération d'initialiser correctement une instance de l'objet APDU et de l'envoyer à la carte via l'instance de la classe CadT1Client.

```

/* Menu principal /////////////////////////////////
boolean fin = false;
while (!fin) {
    System.out.println();
    System.out.println("Application cliente Javacard");
    System.out.println("-----");
    System.out.println();
    System.out.println("1 - Interroger le compteur");
    System.out.println("2 - Incrementer le compteur");
    System.out.println("3 - Decrementer le compteur");
    System.out.println("4 - Reinitialiser le compteur");
    System.out.println("5 - Quitter");
    System.out.println();
    System.out.println("Votre choix ?");
    int choix = 0;
    try {
        choix = System.in.read();
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    while (!(choix >= '1' && choix <= '5')) {
        try {
            choix = System.in.read();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    apdu = new Apdu();
    apdu.command[Apdu.CLA] = Maclasse.CLA_MONAPPLET;
    apdu.command[Apdu.P1] = 0x00;
    apdu.command[Apdu.P2] = 0x00;
    apdu.setLe(0x7f);
    switch (choix) {
        case '1':
            apdu.command[Apdu.INS] = Maclasse.INS_INTERROGER_COMPTEUR;
            cad.exchangeApdu(apdu);
            if (apdu.getStatus() != 0x9000) {
                System.out.println("Erreur : status word different de 0x9000");
            } else {
                System.out.println("Valeur du compteur : " + apdu.dataOut[0]);
            }
            break;
        case '2':
            apdu.command[Apdu.INS] = Maclasse.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);
try {
    cad.exchangeApdu(apdu);
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (CadTransportException e) {
}

```

Etape 4 - Mise hors tension

Éteindre la carte et libérer les ressources pour éviter des erreurs lors d'une utilisation future, cette étape est critique pour terminer le programme proprement

- La déconnexion de la Javacard se fera via la méthode powerDown() de la classe CadT1Client

```

}
/* Mise hors tension de la carte */
try {
cad.powerDown();
} catch (Exception e) {
System.out.println("Erreur lors de l'envoi de la commande Powerdown a la Javacard");
return;
}

```

I.2 Utilisation de l'application cliente avec un simulateur - JCWDE

Afin de pouvoir lancer le simulateur de notre applet en ligne de commande, nous allons créer un fichier "de configuration". Il permet de lister les applets Javacard à installer pour la simulation et de spécifier leurs AID respectifs.

- Créons notre fichier que nous appellerons "monapplet.app" (répertoire parent du package contenant le fichier class de l'applet card « .\workspace\TP Javacard\bin »). Il contiendra la ligne suivante :

```

Fichier   Modifier   Affichage

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

```

eclipse2020 > eclipse > work-space > tpjavacard1 > bin			Rechercher
Nom	Modifié le	Type	
monpackage	15/10/2024 13:59	Dossier de fichiers	
monapplet.app	15/10/2024 00:17	Document texte	

- Nous pouvons maintenant lancer notre simulateur, en prenant soin de nous placer dans le bon répertoire (répertoire parent du package contenant le fichier class de l'applet « .\workspace\TP Javacard\bin »)

```
C:\eclipse2020\eclipse\work-space\tpjavacard1\bin>jcwde monapplet.app.txt
Java Card 2.2.2 Workstation Development Environment, Version 1.3
Copyright 2005 Sun Microsystems, Inc. All rights reserved. Use is subject to license terms.
jcwde is listening for T=1 Apdu's on TCP/IP port 9?025.
```

- Maintenant que notre simulateur est lancé, lançons notre application cliente :

```
Application cliente Javacard
-----
1 - Interroger le compteur
2 - Inrementer le compteur
3 - Decrementer le compteur
4 - Reinitialiser le compteur
5 - Quitter

Votre choix ?
```

- Commençons par interroger le compteur : tapons 1 puis validons :

```
Sending command: CLA=0 INS=-92 P1=4 P2=0 Data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 0]

Application cliente Javacard
-----
1 - Interroger le compteur
2 - Inrementer le compteur
3 - Decrementer le compteur
4 - Reinitialiser le compteur
5 - Quitter

Votre choix ?

1
Valeur du compteur : 0
```

- Le compteur est bien à 0. Incrémentation maintenant le compteur 3 fois puis décrémentons-le une fois. Lorsque nous réinterrogeons la carte, le compteur vaut 2 donc tout va bien :

```
Sending command: CLA=0 INS=-92 P1=4 P2=0 Data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 0]

Application cliente Javacard
-----
1 - Interroger le compteur
2 - Inrementer le compteur
3 - Decrementer le compteur
4 - Reinitialiser le compteur
5 - Quitter

Votre choix ?
1
Valeur du compteur : 0

Application cliente Javacard
-----
1 - Interroger le compteur
2 - Inrementer le compteur
3 - Decrementer le compteur
4 - Reinitialiser le compteur
5 - Quitter

Votre choix ?
2
OK

Votre choix ?

3
OK

Application cliente Javacard
-----
1 - Interroger le compteur
2 - Inrementer le compteur
3 - Decrementer le compteur
4 - Reinitialiser le compteur
5 - Quitter

Votre choix ?
```

```
Application cliente Javacard
```

- ```

1 - Interroger le compteur
2 - Inrementer le compteur
3 - Decrementer le compteur
4 - Reinitialiser le compteur
5 - Quitter
```

```
Votre choix ?
```

```
1
```

```
Valeur du compteur : 2
```

- Quittons maintenant notre application cliente (commande 5). Nous pouvons voir que
- le simulateur se termine automatiquement à la réception de la commande
- "powerdown" :

```
C:\eclipse2020\eclipse\work-space\tpjavacard1\bin>jcwde monapplet.app.txt
Java Card 2.2.2 Workstation Development Environment, Version 1.3
Copyright 2005 Sun Microsystems, Inc. All rights reserved. Use is subject to license terms.
jcwde is listening for T=1 Apdu's on TCP/IP port 9025.
jcwde exiting on receipt of power down command.
```

### Remarque

Le simulateur JCWDE ne permet pas de conserver l'état de la carte. Si nous le relançons, le compteur sera à 0 et non à 2.

```
Sending command: CLA=0 INS=-92 P1=4 P2=0 Data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 0]
```

```
Application cliente Javacard
```

- ```
-----  
1 - Interroger le compteur  
2 - Inrementer le compteur  
3 - Decrementer le compteur  
4 - Reinitialiser le compteur  
5 - Quitter
```

```
Votre choix ?
```

```
1
```

```
Valeur du compteur : 0
```

2024 - 2025

Mini projet : Distributeur JavaCard



Réalisé par : Sirin GRIRA & Syrine AMMAR & Roua BEN TIBA,
ING1 INFO

But de ce projet

L'objectif de ce projet est d'intégrer une interface graphique (GUI) construite à l'aide de Java Swing avec une application JavaCard.

Ce projet montre comment envoyer et recevoir des transactions sécurisées entre une application cliente (développée avec Java Swing et NetBeans) et un applet JavaCard.

Outils et Technologies Utilisés

- **Java Swing** : Utilisé pour créer l'interface graphique (GUI) de l'application cliente.
- **NetBeans IDE** : Utilisé comme environnement de développement intégré (IDE) pour concevoir l'interface graphique et coder l'application cliente.
- **JavaCard** : Une plateforme pour développer des applications sécurisées pour les cartes à puce.
- **APDU (Application Protocol Data Unit)** : Le protocole de communication utilisé entre le client et la JavaCard.
- **Programmation Socket** : Utilisée pour établir la communication entre l'application cliente et la JavaCard via TCP/IP.

Architecture du Système

Le système se compose de deux composants principaux :

1. Côté Client (GUI Swing) :

- L'utilisateur interagit avec une interface graphique où il peut effectuer des opérations telles que **déposer** ou **retirer** de l'argent.
- Le client communique avec l'application JavaCard côté serveur en utilisant des **sockets** et des commandes **APDU**.

2. Côté Serveur (JavaCard) :

- L'application JavaCard côté serveur traite les **transactions** sécurisées et renvoie les **résultats**.
- Elle utilise des commandes **APDU** pour interagir avec le client.
- L'application cliente **envoie** des commandes APDU à la JavaCard, où l'applet JavaCard traite les commandes et **renvoie** une réponse.

Explication du Code de l'Applet JavaCard et de la Partie Client

1. Partie Applet JavaCard

a. Déclaration des constantes et variables :

- Des constantes sont définies pour identifier l'applet et les commandes spécifiques (ex. **CLA_MONAPPLET** pour l'applet et **INS_TEST_CODE_PIN** pour tester le code PIN) pour la gestion du solde, du code PIN, et de l'historique des transactions.

b. Initialisation de l'applet (install) :

- La méthode **install** est utilisée pour créer l'applet et l'enregistrer sur la carte. Cette méthode est appelée lors de l'installation de l'applet sur la carte.

c. Sélection de l'applet (select) :

- Lorsqu'un terminal **sélectionne** l'applet, la méthode **select** est appelée. Elle vérifie si le nombre de tentatives d'entrée du code PIN a été dépassé, ce qui empêcherait la sélection de l'applet.

d. Traitement des commandes (process) :

- La méthode **process** traite les commandes reçues via APDU. Elle vérifie les instructions envoyées, comme la vérification du PIN ou l'ajout d'argent, et exécute l'action appropriée en fonction de la commande.

e. Vérification du PIN (VerificationPIN) :

- La méthode vérifie si le code PIN fourni par l'utilisateur correspond au **PIN stocké**. Si ce n'est pas le cas, une erreur est retournée, limitant le nombre de tentatives possibles.

f. Gestion des opérations sur le compte :

- **AjouterArgent** permet d'ajouter de l'argent au solde après validation du PIN et en s'assurant que le montant ajouté respecte les limites fixées.
- **ExtraireArgent** permet de retirer de l'argent du solde si celui-ci est suffisant pour couvrir le retrait demandé.

g. Consultation du solde (getBalance) :

- Cette méthode renvoie le **solde du compte** sous forme d'octets. Elle divise le solde en plusieurs parties si nécessaire pour respecter les limites de taille des APDU.

h. Historique des transactions (afficherHistorique) :

- Cette méthode permet de récupérer l'historique des transactions effectuées sur le compte, comme les ajouts et les retraits d'argent, et de le renvoyer à l'application cliente.

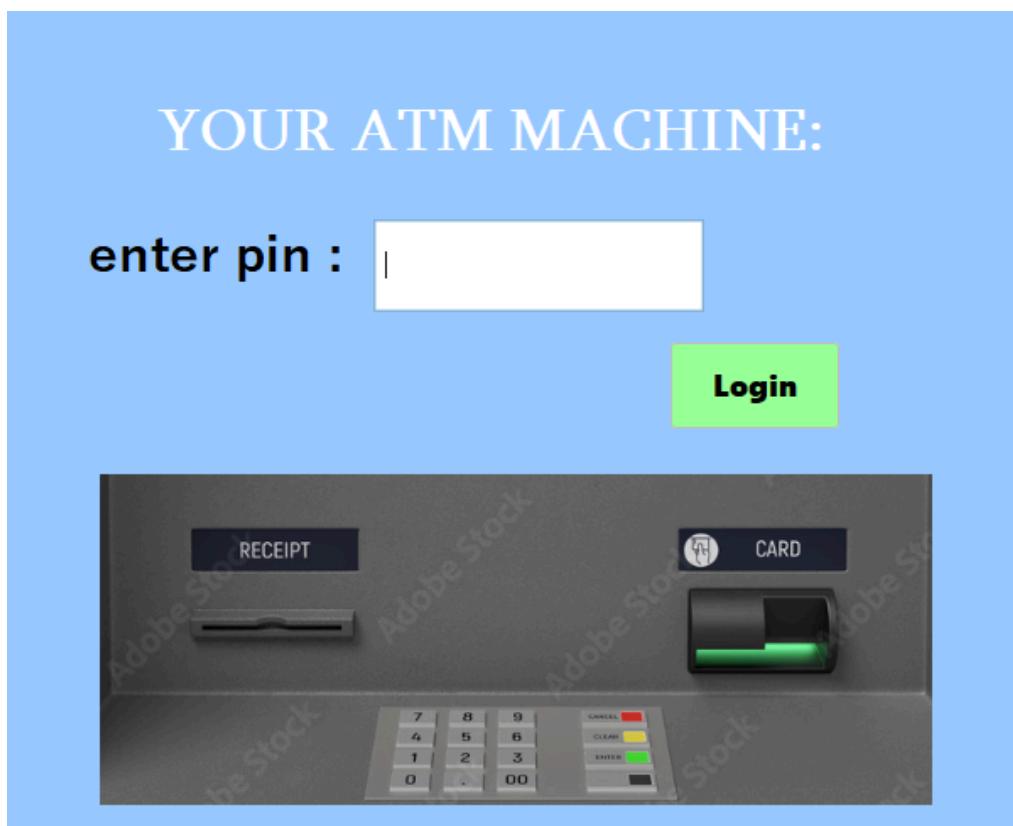
2. Partie Client

a. Préparation de l'interface graphique

La préparation de l'interface graphique du projet a été réalisée en utilisant Java Swing, permettant de concevoir une interface utilisateur interactive.

b. Les différentes interfaces de notre application

Au moment d'exécution, cette interface ATM s'affiche et on doit taper le code PIN :



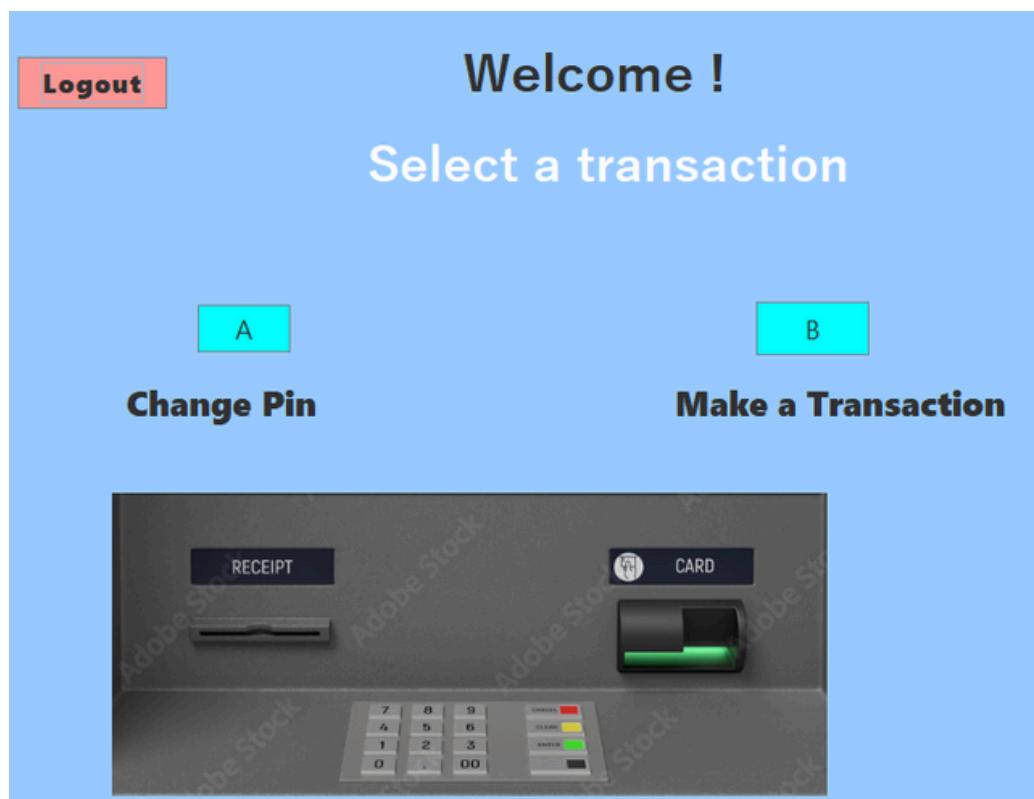
Si le code saisis est incorrect, une alert s'affiche :



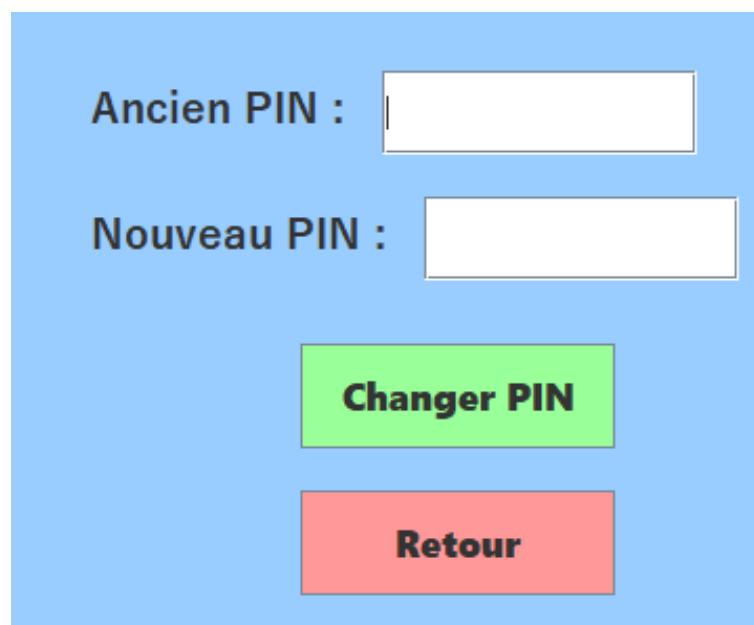
La même chose si on saisit plus que 4 chiffres :



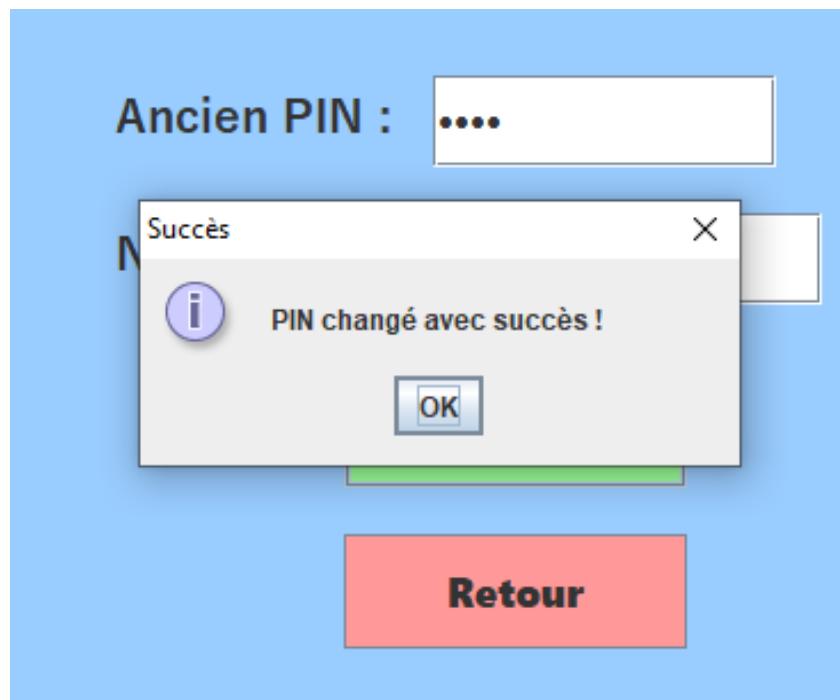
Ensute, l'interface MainPanel est appelée dont on peut choisir une opération :



Voici l'interface Change PIN qui nous permet de changer le code PIN de la Carte :



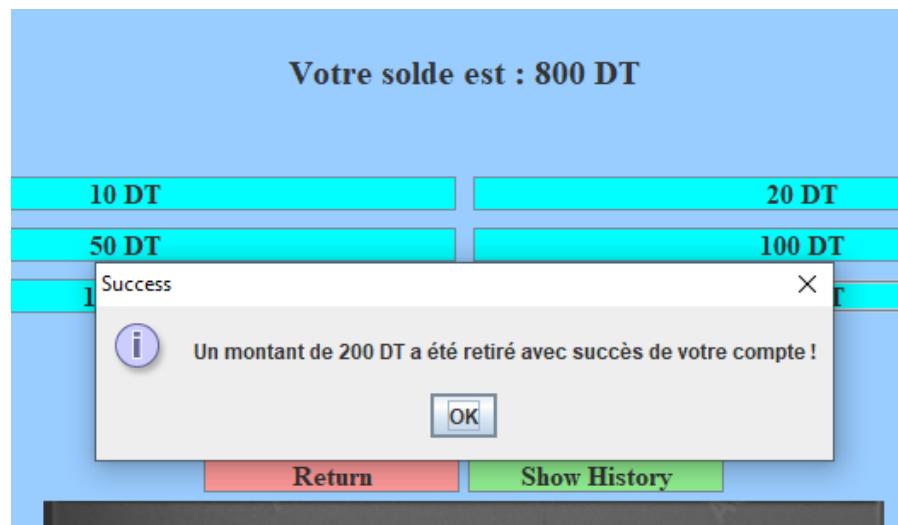
Au cas de succès, ce message s'affiche



Ainsi, voici l'interface MakeTransaction qui nous permet de créditer (ou débiter) un tel montant :



Au cas de succès de transaction, ce message s'affiche



Et Voici l'interface Historique qui nous permet de consulter les transactions récentes :

