**Institut Supérieur d’Informatique et de Mathématiques de Monastir**



**Niveau : ING1 INFO**

**Groupe : TP1**

**Réalisé par :**

**Fares Aloulou – Rayen Braiek – Mahmoud Ben Abdelkader**

**Enseignant : Mr Ramzi Mahmoudi**

Table des matières

Table des matières iii

Chapitre 1. Environnement de développement JavaCard 3

I.1 But 3

I.2 Téléchargement des fichiers  3

I.3 Installation des programmes  4

### I.3.1java développement kit (jdk)…………………………………………………………………….4

I.3.2java card development kit (jcdk)………………………………………………………………...4

I.3.3Réglage des variables d’environnement ………………………………………………………. 5

I.3.4Installation Eclipse……………………………………………………………………………... 7

I.4 Conclusion  8

Chapitre II. Développement d’une application coté Serveur 8

II.1 Les différentes étapes de développement 8

II.2 Codage de notre applet 11

II.3 Outils de simulation : 15

Chapitre III. Programmation d’une application coté client 17

III.1 Création de l’application client sous Eclipse : 17

III.2 Utilisation de l'application cliente avec un simulateur – JCWDE : 21

Chapitre IV. Réalisation du Mini-Projet 23

IV.1 Partie Serveur   23

IV.1.1 Déclaration des variables et des constantes 23

IV.1.2 Implémentations des méthodes nécessaires  24

IV.2 Partie Client  26

IV.2.1 Les interfaces de notre application Client  26

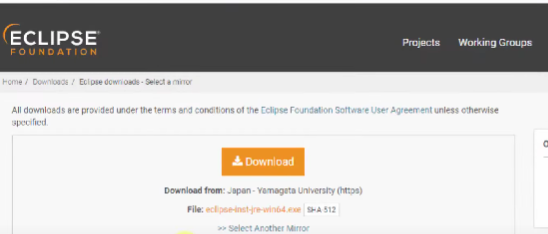
Chapitre I.

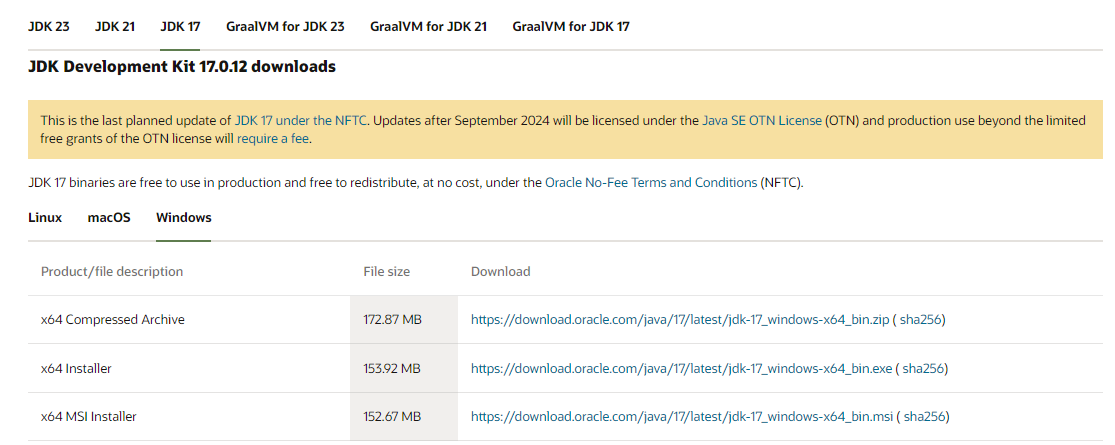
Environnement de développement JavaCard

1. *But :*

L'objectif de ce TP est d'installer l'environnement de développement requis pour créer des applications JavaCard. Nous allons procéder à l'installation d'Eclipse, l'IDE (Environnement de Développement Intégré), de la plateforme JavaCard 2.2.2 (kit de développement) et du plug-in Eclipse-JCDE (interface reliant la plateforme JavaCard à Eclipse).

1. *Téléchargement des fichiers :*

**

**

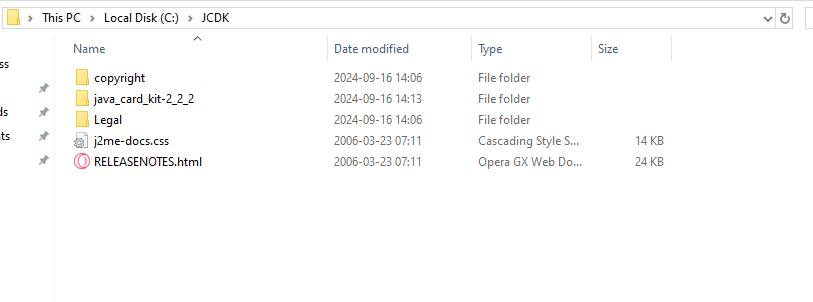
*3.Installation des programmes :*

*3.1- Java développement kit (jdk) :*

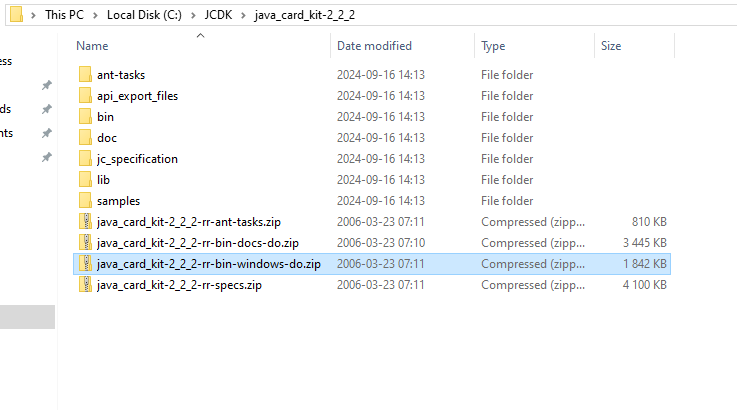
Après avoir exécuté le fichier d’installation de jdk17 on transfert les fichiers vers un répertoire plus accessible comme **‘’ C:\java ‘’**

*3.2-java card development kit (jcdk):*

On Décompresse l’archive ‘’java\_card\_kit-2\_2\_2-windows.zip’’ dans le répertoire **‘’C:\JCDK.”**

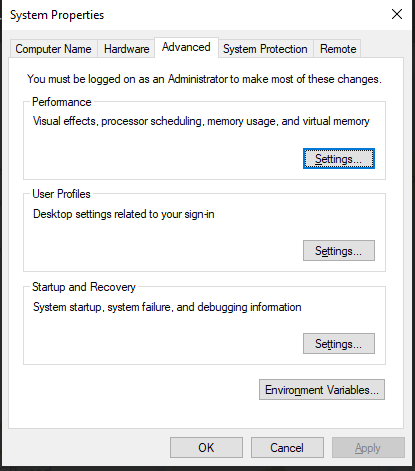


Dans le répertoire **“ C:\JCDK \java\_card\_kit-2\_2\_2 ‘**’ on doit décompresser les fichiers valables jusqu’ à avoir l'arborescence suivante :

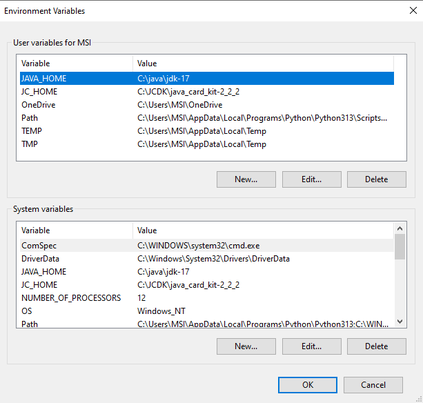
****

*3.3-Réglage des variables d’environnement :*

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

****

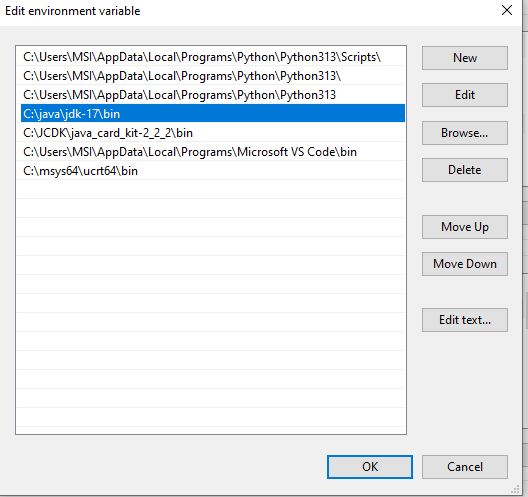
On ajoute les variable d’environnement dans l’espace utilisateur :  
**JAVA\_HOME=‘’C:\java‘’  
JC\_HOME = ‘’ C:\JCDK \java\_card\_kit-2\_2\_2 ‘’**

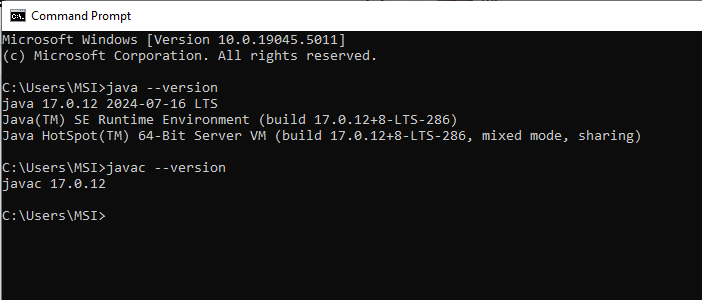
****

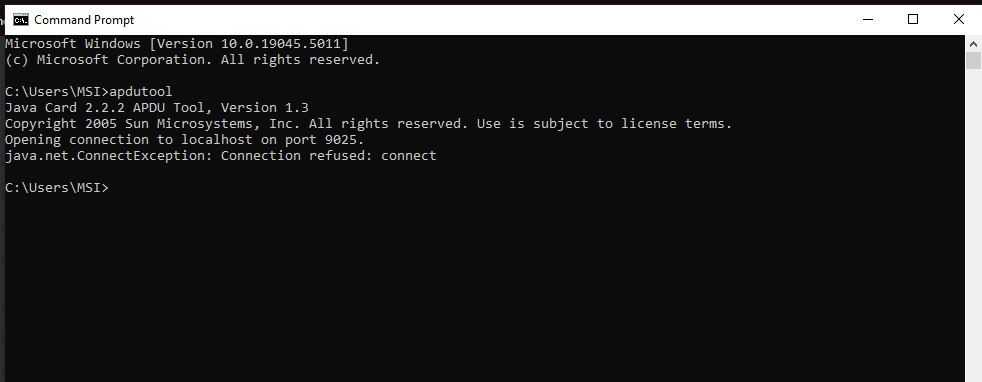
On sélectionne la ligne Path puis faire Modifier la variable Path ,

Cette interface apparaitra,ou on va ajouter les références :  
**’%JAVA\_HOME%\bin’  
‘%JC\_HOME\bin% ‘**

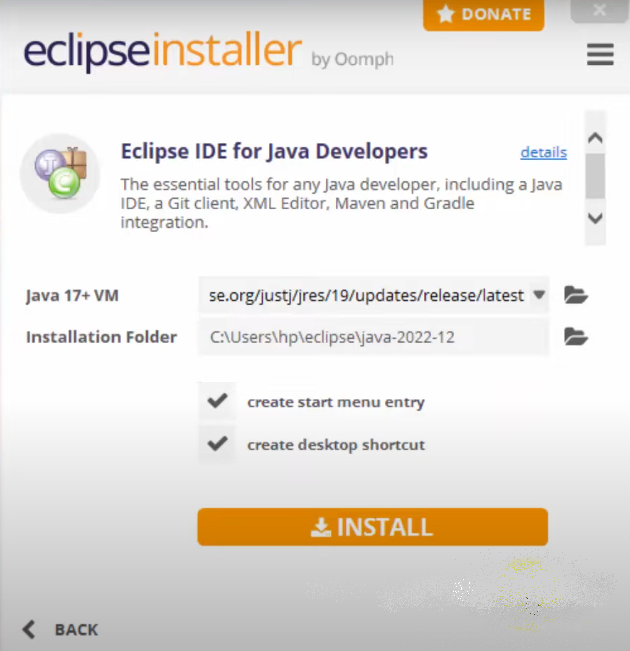


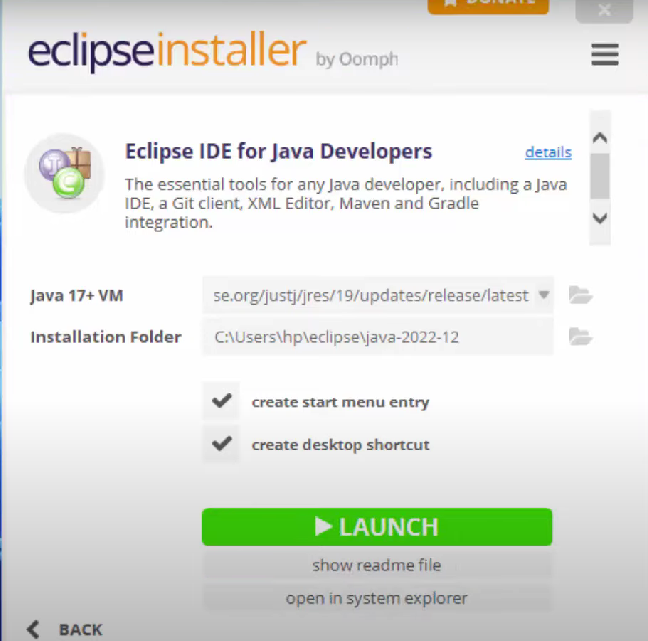
****





*3.4 Installation Eclipse :*





Maintenant, il faut mettre à jours le plugin jcde de l’eclipse. Il sagit de décompresser l’archive eclipse-jcde-0.2.zip après l’avoir téléchargée dans le répertoire **‘’C:\Eclipse\eclipse\plugins’’** Un sous-dossier "plugins " sera créer, il faut simplement recopier tout son contenu directement sous **C:\Eclipse\eclipse\plugins.**

*4.Conclusion :*

A ce stade, nous disposons d'un environnement de développement complet permettant de créer des Applet Javacard, de les simuler et de créer des applications clientes.

Chapitre II.

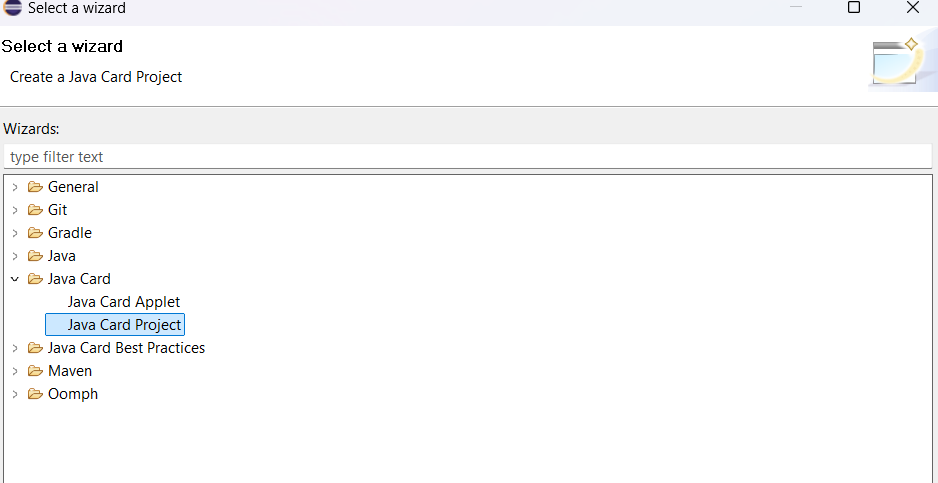
Développement d’une application coté Serveur

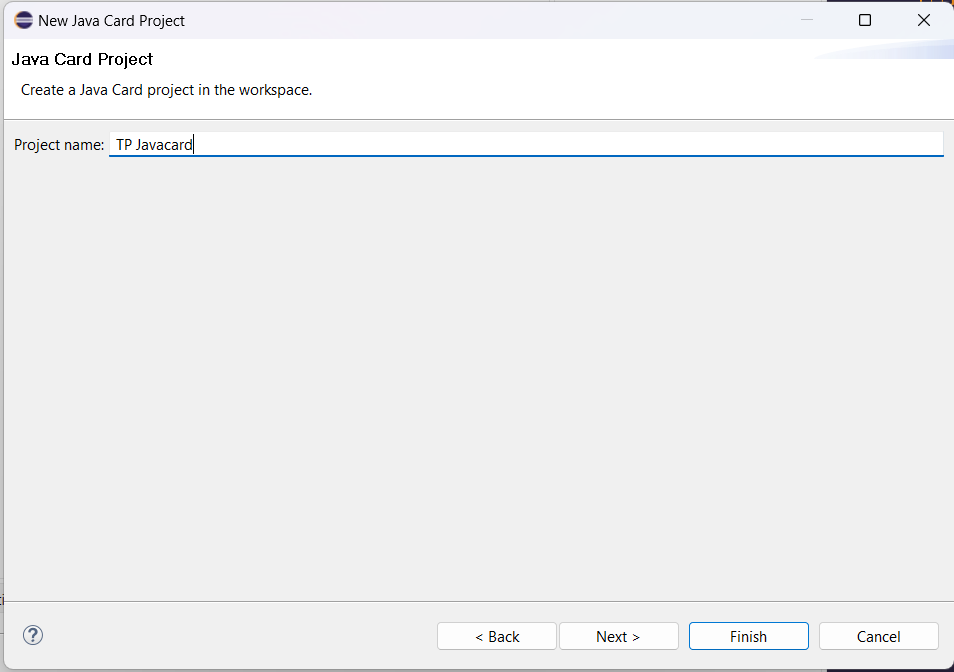
## Les différentes étapes de développement :

## I.1 Programmation de l'application Serveur :

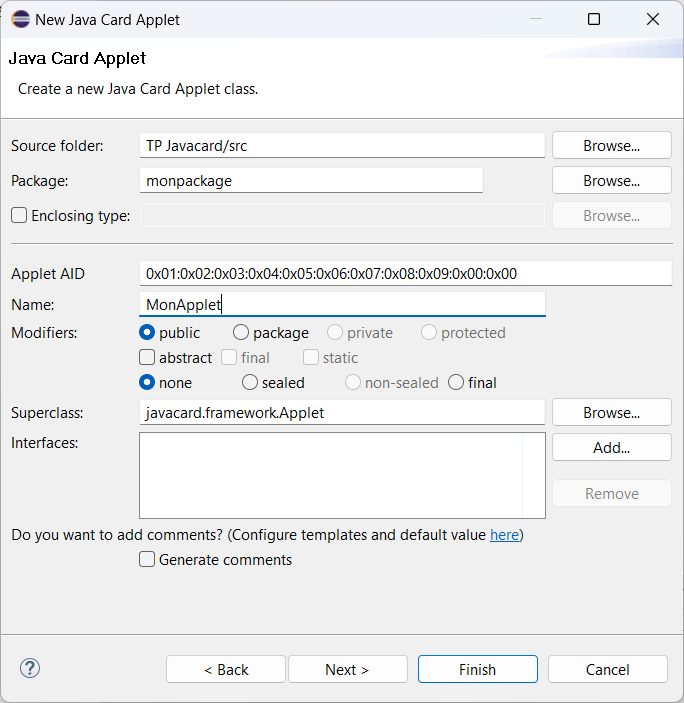
## Création de l'applet card sous Eclipse :

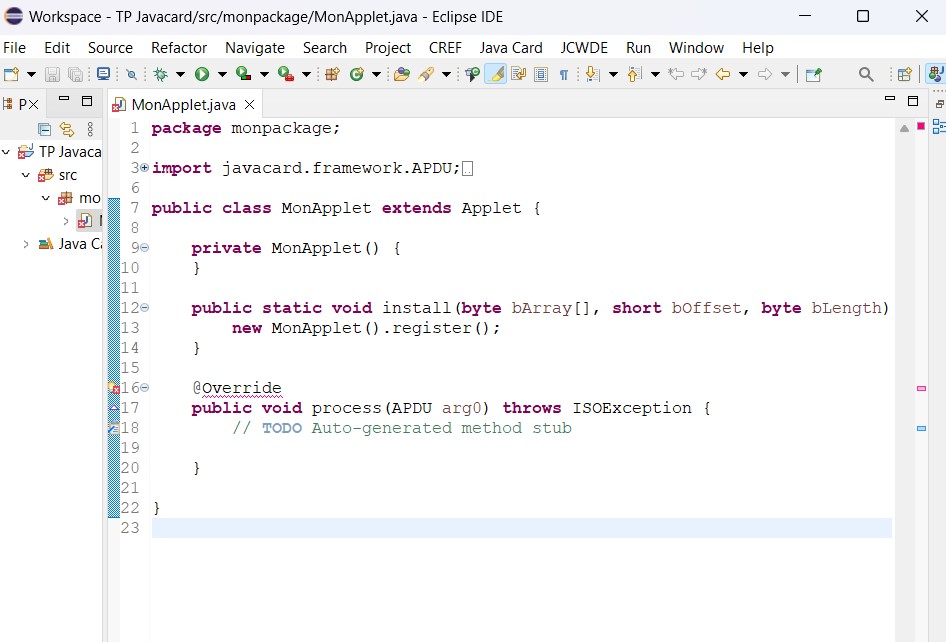
File>New > Other >"Java Card Project"



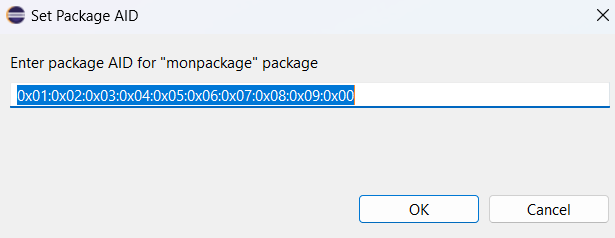


File>New > Other >"Java Card applet"





Clic droit sur le package monpackage >Java Card Tools> Set Package AID :



## I.2 Codage de notre applet :

Nous allons développer une applet de compteur qui offrira quatre fonctionnalités : augmenter la valeur du compteur,

diminuer la valeur du compteur,

consulter la valeur actuelle du compteur, réinitialiser le compteur à une valeur spéciﬁque.

### Étape 1 : Intégration de l'API JavaCard

L'API Java Card propose un ensemble de classes dédiées à la programmation d'applications pour cartes intelligentes, conformément à la norme ISO 7816. Elle contient un package

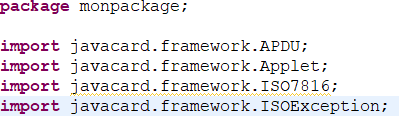
principal, javacard.framework, pour le développement d'une applet pour cartes, et offre les éléments suivants :

La classe javacard.framework.Applet, qui est la superclasse pour les applets résidant sur la carte. Une applet Java Card doit hériter de cette classe.

La classe javacard.framework.APDU, qui encapsule les commandes APDU.

L'interface javacard.framework.ISO7816, qui déﬁnit un ensemble de mots d'état pour signaler les erreurs courantes dans les applets.

La classe javacard.framework.ISOException, utilisée pour gérer les exceptions.



### Étape 2 : Déclaration des attributs et des constantes

Dans cette étape, nous allons déﬁnir les éléments suivants :

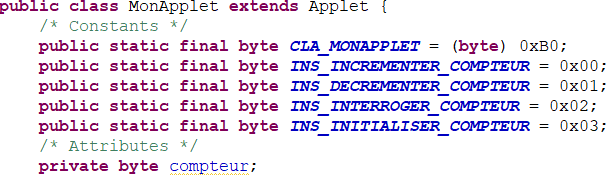
**Constantes** (de type primitif byte):

CLA\_MONAPPLET : Identiﬁe la classe des commandes APDU prises en charge par notre applet JavaCard.

Codes INS : Déﬁnissent le type d'instruction des commandes APDU traitées par l'applet.

**Attributs** (de type primitif byte) :

Compteur : Déﬁnit la variable qui sera utilisée pour l'incrémentation ou la décrémentation.



### Étape 3 : Déflnition des méthodes publiques à implémenter

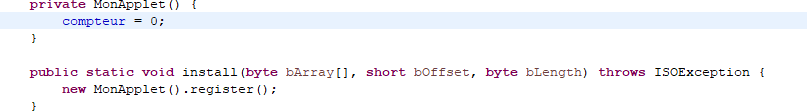
Nous allons déﬁnir les méthodes obligatoires pour l'applet, en commençant par :

1. **La méthode install()** : création et enregistrement de l'instance Applet

L'applet doit implémenter la méthode install() pour créer une instance de l'applet et l'enregistrer dans le JCRE en appelant la méthode register(). La méthode install() prend en paramètre un vecteur d'octets qui contient les paramètres nécessaires pour initialiser et personnaliser l'instance de l'applet lors de l'installation.

Une fois installée, l’applet Java Card reste en veille tant qu’elle n’est pas explicitement

sélectionnée. La classe javacard.framework.Applet propose une implémentation par défaut pour les méthodes select() et deselect().



### La méthode process() : Traitement des commandes APDU

Une fois l’applet sélectionnée, le JCRE dirige toutes les commandes APDU (y compris la commande SELECT) vers la méthode process() de l'applet. Cette méthode permet à

l’applet de décoder chaque commande APDU et d'exécuter l’action indiquée. Pour chaque commande traitée, l'applet répond en renvoyant le résultat.

L’applet gère une commande APDU en utilisant des méthodes sur l’objet APDU et suit généralement les étapes suivantes :

### Extraction du buffer APDU :

L’applet appelle la méthode getBuffer() pour obtenir une référence au buffer APDU, qui contient le message sous forme de tableau d’octets.

byte[] buffer = apdu.getBuffer();

### Vérification des octets d'en-tête de l'APDU :

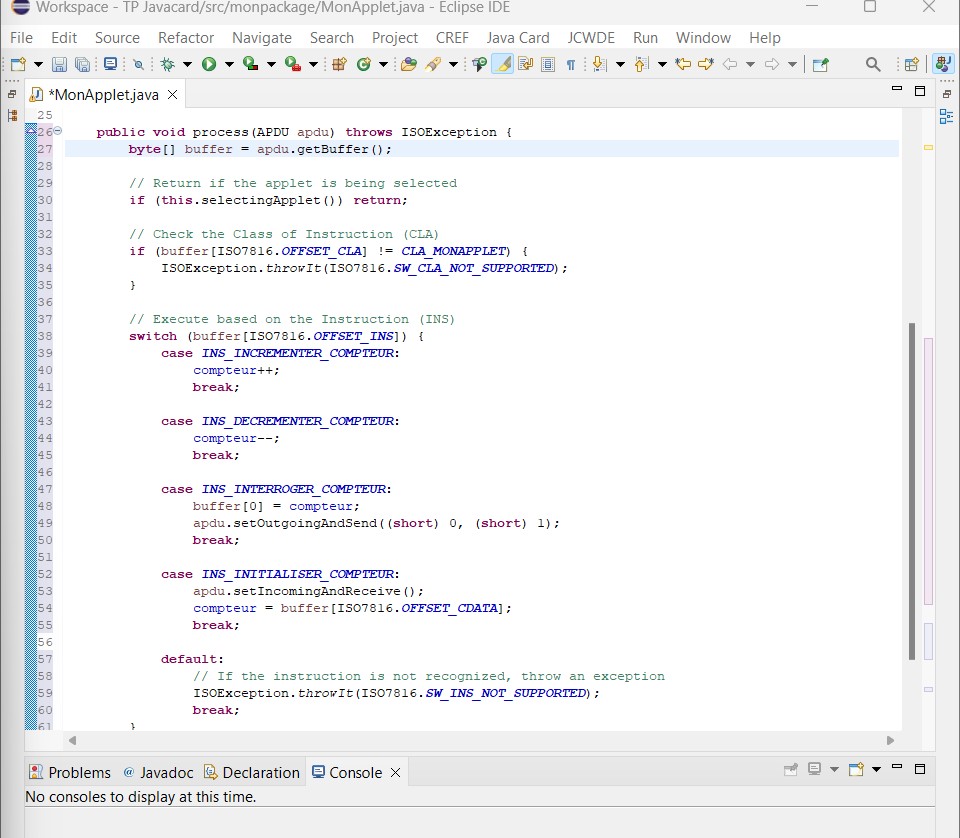
Lorsque l’applet reçoit l’objet APDU, seuls les cinq premiers octets du buffer sont disponibles

: CLA, INS, P1, P2, et P3. L'octet P3 représente Lc si la commande contient des données optionnelles. L’applet peut examiner les octets d'en-tête pour déterminer :

la structure de la commande (buffer[ISO7816.OFFSET\_CLA]),

l’instruction spéciﬁque de la commande (buffer[ISO7816.OFFSET\_INS]).

La méthode selectingApplet() permet de différencier la commande APDU SELECT qui

sélectionne l’applet de celles qui sont utilisées pour sélectionner des ﬁchiers ou modiﬁer l'état interne de l'applet.

Si la commande APDU contient des données optionnelles (par exemple, pour initialiser le compteur), l’applet doit orienter l’objet APDU vers la réception de ces données en appelant la méthode setIncomingAndReceive().

Après avoir traité une commande APDU, l’applet peut renvoyer des données à

l’application sous forme de réponse APDU en utilisant la méthode setOutgoingAndSend(). La norme exige que ce buffer ait une capacité minimale de 133 octets (5 octets pour l'en- tête + 128 octets pour les données).

### d. Renvoyer le mot d’état (status word) :

Après un traitement réussi dans la méthode process(), le JCRE renvoie automatiquement le mot d’état 0x9000 pour indiquer un traitement normal. Si, à tout moment, l’applet détecte une erreur, elle peut lever une exception en appelant la méthode statique ISOException.



Voici une reformulation :

## I.3 Outils de simulation :

Avant de commencer les simulations, effectuez un clic droit sur le package monpackage,

sélectionnez **Java Card Tools**, puis **Generate Script**. Cela génère automatiquement les APDU nécessaires pour le téléchargement, l'installation (instanciation) et la sélection de l'applet sur une JavaCard. Le package monpackage.javacard contient alors trois scripts :

cap-download.script : téléchargement de l'applet create-MonApplet.script : installation de l'applet select-MonApplet.script : sélection de l'applet

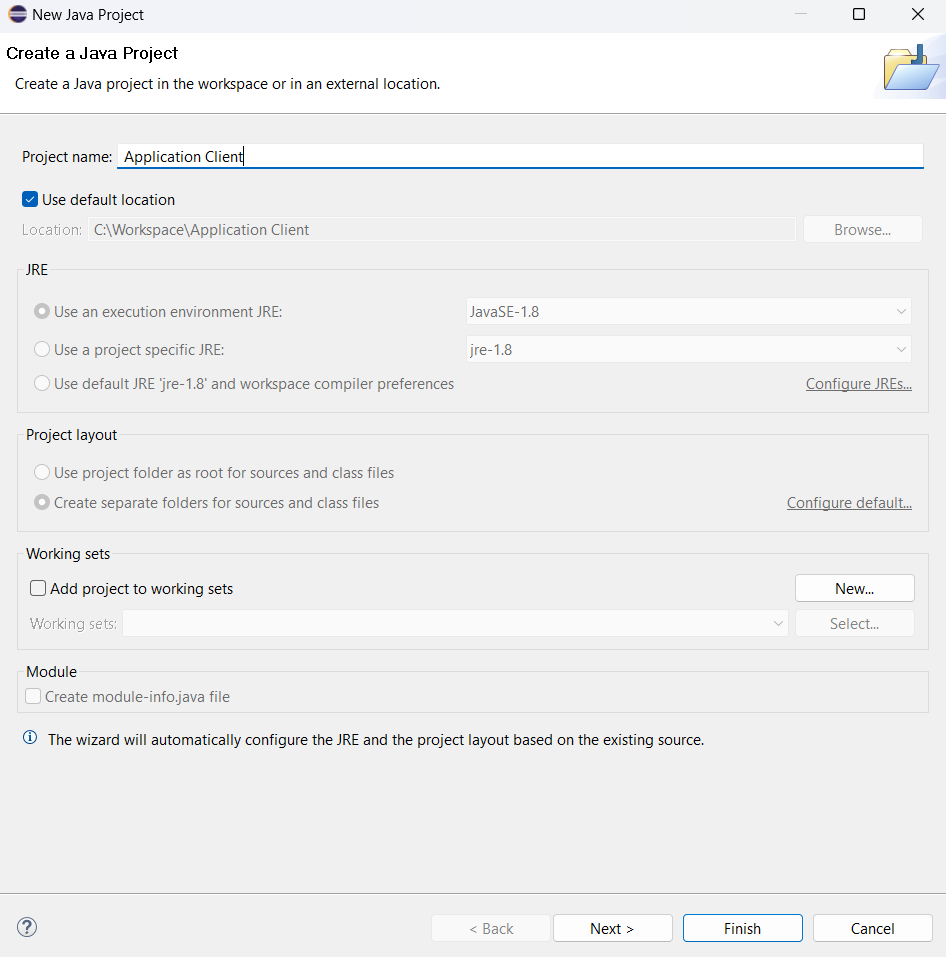
### 

Chapitre III.

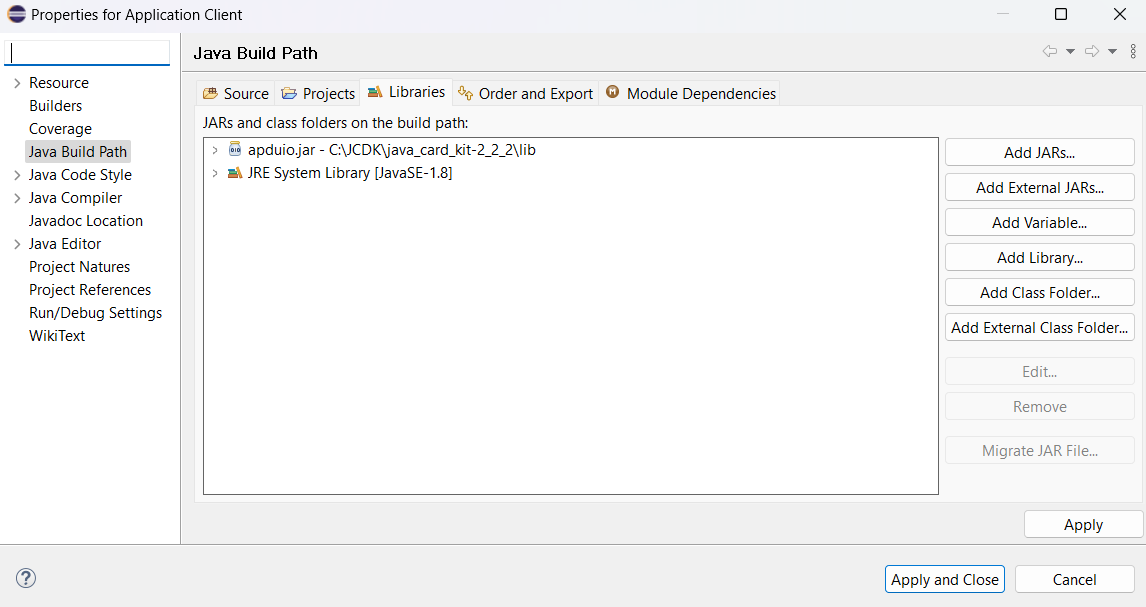
Programmation d’une application coté client

I.1 Création de l’application client sous Eclipse :

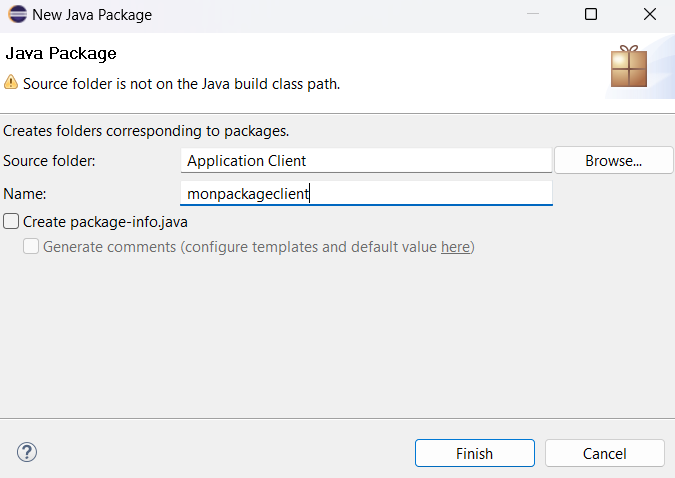
a) Création d’un nouveau projet :



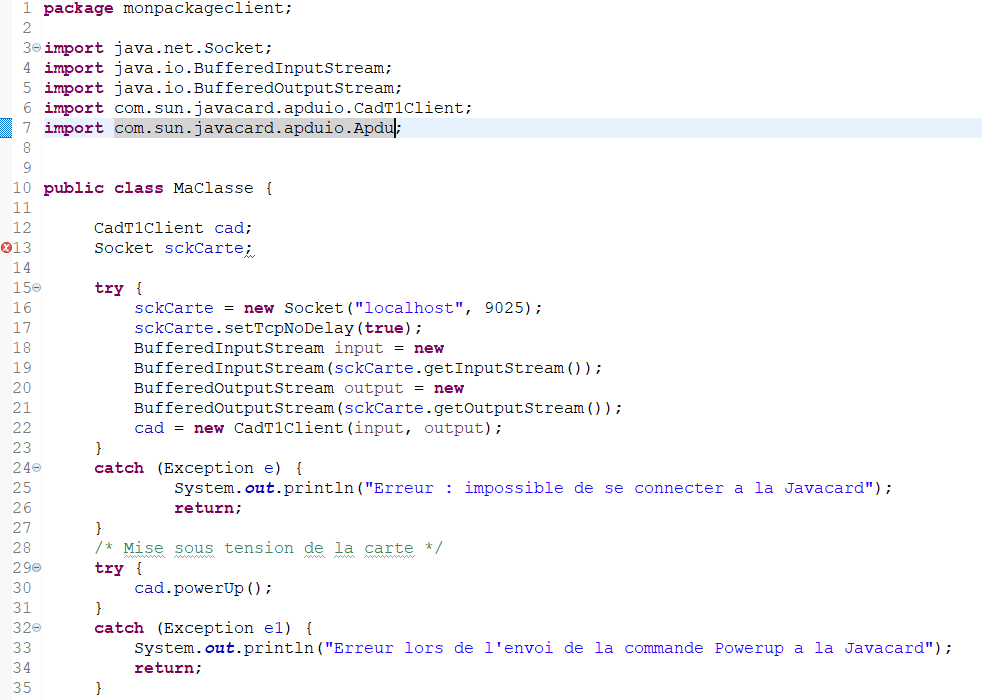
b) Ajout de la librairie « apduio » dans le classpath :



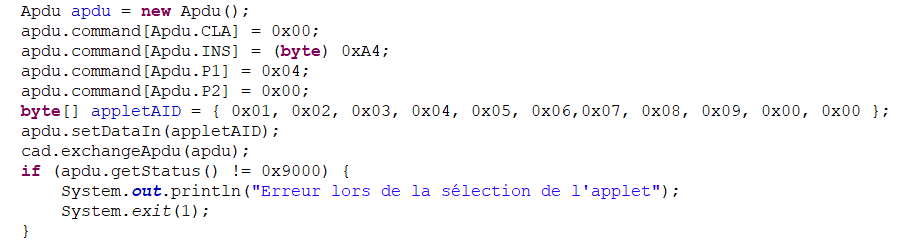
c) Création de la classe principale :



Etape 1 - Connexion :

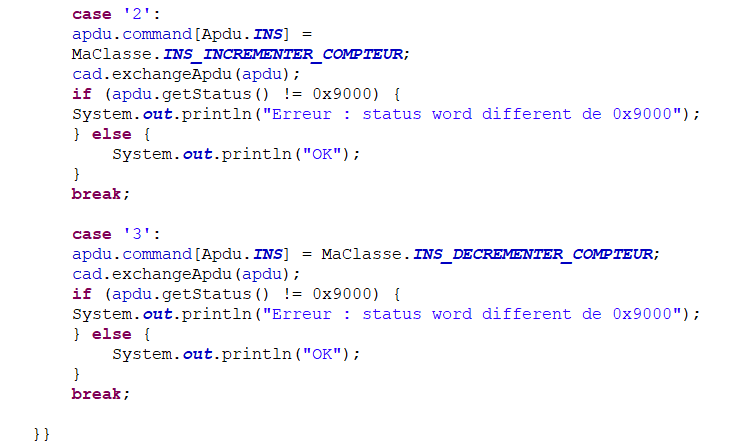


Etape 2 – Sélection :

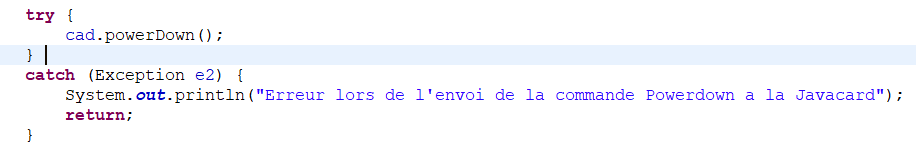


Etape 3 - Invocation des services implémentés :

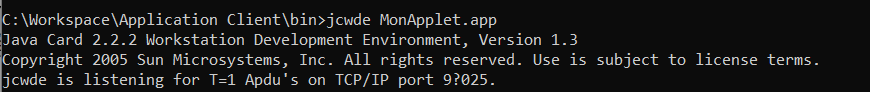




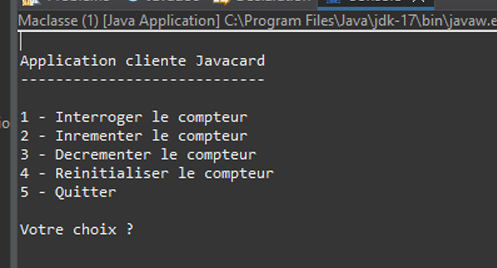
Etape 4 - Mise hors tension :

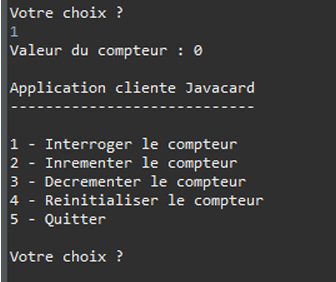


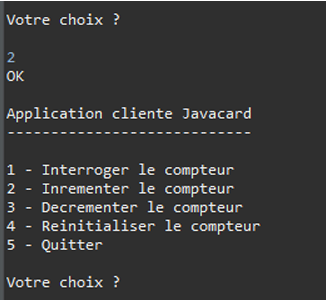
I.2 Utilisation de l'application cliente avec un simulateur – JCWDE :

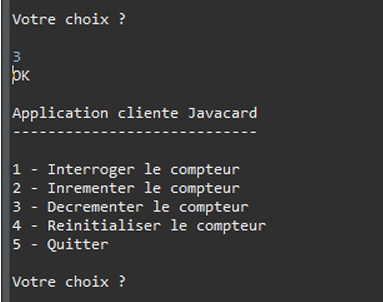


Lancement l’application cliente :









Quittons maintenant notre application cliente :



Capture15.png

Chapitre IV.   
Réalisation du Mini-Projet

* 1. Partie Serveur :
     1. Déclaration des variables et des constantes :

/\* Constants \*/

**Public static final byte *CLA\_MONAPPLET*** = (**byte**) 0xB0;

**Public static final byte *INS\_TEST\_CODE\_PIN*** = 0x00;

**Public static final byte *INS\_INTERROGER\_COMPTE*** = 0x01;

**Public static final byte *INS\_INCREMENTER\_COMPTE*** = 0x02;

**Public staticf inal byte *INS\_DECREMENTER\_COMPTE*** = 0x03;

**Public staticf inal byte *INS\_INITIALISER\_COMPTE*** = 0x04;

**Public final static short *MAX\_BALANCE*** = 0x01F4;

// le maximum dela balance = 500 TND

**Public final static byte *MAX\_MONTANT\_TRANSACTION*** = (**byte**)127;

// maximum montantqu'on peuttransiter

**Public final static byte *MAX\_ERROR\_PIN*** = (**byte**) 0x03;

// maximum de code pinerroner

**Public final static byte *MAX\_PIN\_LENGTH*** = (**byte**) 0x04;

// longeurmaximaledu code pin

**Private byte**[] INIT\_PIN = { (**byte**) 2, (**byte**) 0,(**byte**) 0,(**byte**) 1 };

/\* Exception \*/

// Verification Pin Echoué

**Final static short *SW\_VERIFICATION\_FAILED*** = 0x6300;

**Final static short *SW\_EXCEED\_TRY\_LIMIT*** = 0x6321;

// signal the the PIN validation is required

// for a credit or a debit transaction

// signal that the balance exceed the maximum

**Final static short *SW\_EXCEED\_MAXIMUM\_BALANCE*** = 0x6A84;

// signal the the balance becomes negative

**Final static short *SW\_NEGATIVE\_BALANCE*** = 0x6A85;

/\* variables \*/

OwnerPIN pin;

**Static short** *balance* ;

* + 1. Implémentations des méthodes nécessaires :

**Constructeur :**

**private** Bank(**byte**[] bArray,**int** i,**int** j) {

pin = **new** OwnerPIN(***MAX\_ERROR\_PIN***, ***MAX\_PIN\_LENGTH***);

// Initialization parametre pin

pin.update(INIT\_PIN,(**short**) 0, (**byte**) 0x04);

}

**Redéfinitions de la méthode install() :**

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

**new** Bank(bArray,bOffset,bLength).register();

}

**Redéfinitions de la méthode select() :**

**publicboolean** select() {

// pas de selection sile pin est blocker

**if** (pin.getTriesRemaining() == 0)

**returnfalse**;

**returntrue**;

}

**Redéfinitions de la méthode deselect() :**

**publicvoid** deselect() {

pin.reset();

}

**Redéfinitions de la méthode process () :**

**Public void** process(APDU apdu) {

**byte**[] buffer = apdu.getBuffer();

// exception qui teste sur la commande de selection

**if** (apdu.isISOInterindustryCLA()) {

**if** (buffer[ISO7816.***OFFSET\_INS***] == (**byte**) (0xA4)) {

**return**;

} **else** {

ISOException.*throwIt*(ISO7816.***SW\_CLA\_NOT\_SUPPORTED***);

}

}

**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\_TEST\_CODE\_PIN***:

verify(apdu);

**break**;

**case*INS\_INCREMENTER\_COMPTE***:

credit(apdu);

**break**;

**case*INS\_DECREMENTER\_COMPTE***:

debit(apdu);

**break**;

**case*INS\_INTERROGER\_COMPTE***:

getBalance(apdu);

**break**;

**default**:

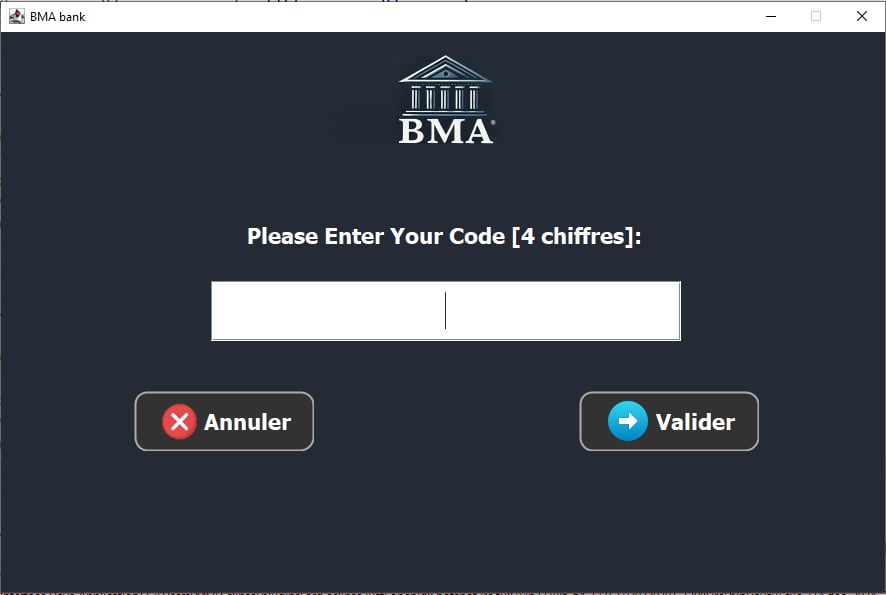
ISOException.*throwIt*(ISO7816.***SW\_INS\_NOT\_SUPPORTED***);

}

}

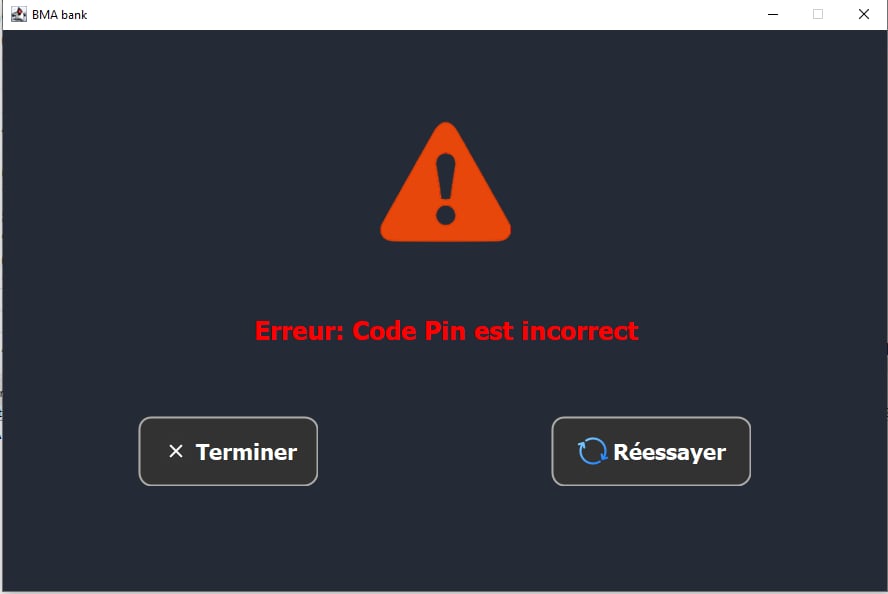
* 1. Partie Client :
     1. Les interfaces de notre application Client :

La première interface qui s’affiche lors de l’exécution. Il faut saisir le code pin dans la zone du Texte puis valider. Un apdu sera envoyer au Serveur pour confirmer si le code pin est correct ou non.



**1ére Cas :**

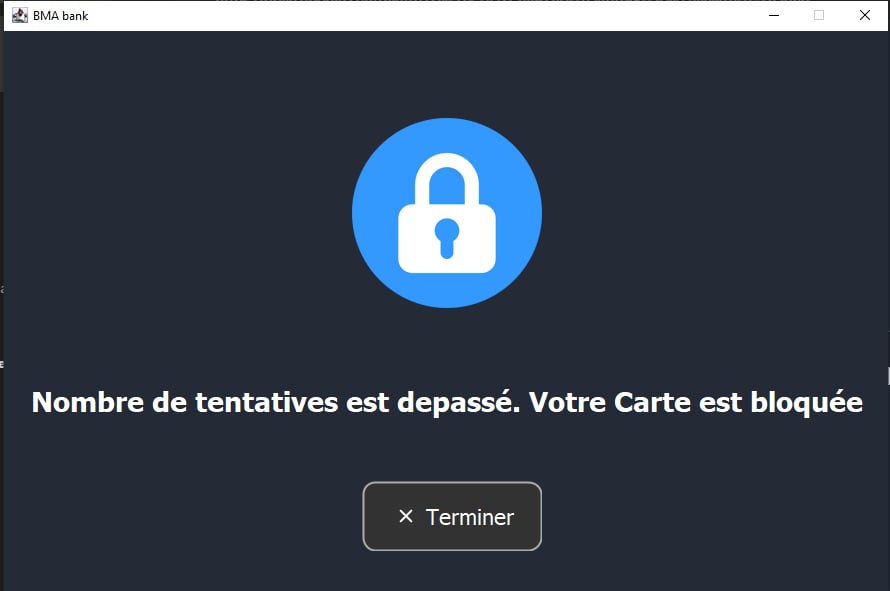
Si le code Pin saisi est **incorrect** alors une fenêtre d’erreur s’affiche.



* Vous pouvez cliquer sur **Réessayer** pour ressaisir votre code PIN ou bien **Terminer** pour fermer le programme.

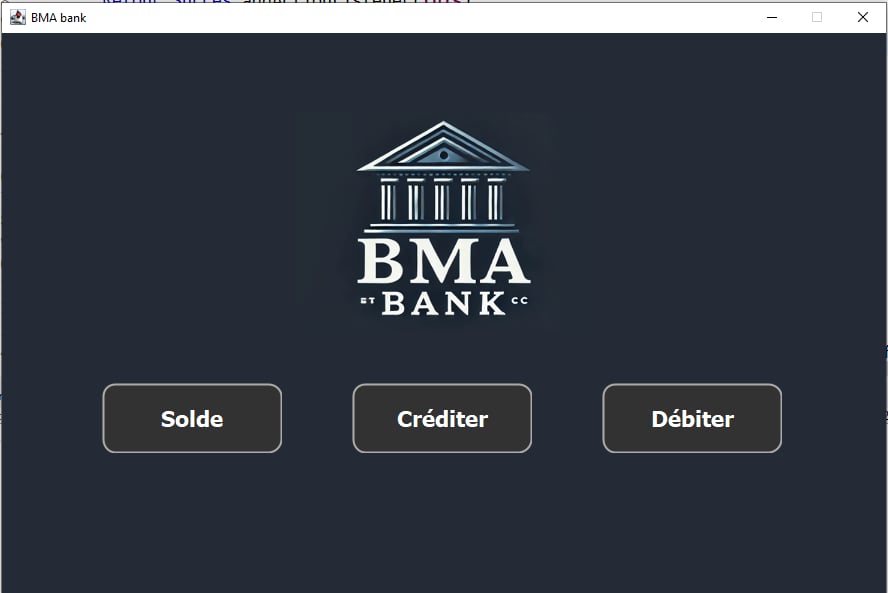
**2éme Cas :**

Si vous avez saisi un code PIN incorrect 3 fois, votre carte sera bloquée.



**3éme Cas :**

Si vous avez saisi le code PIN **correctement**



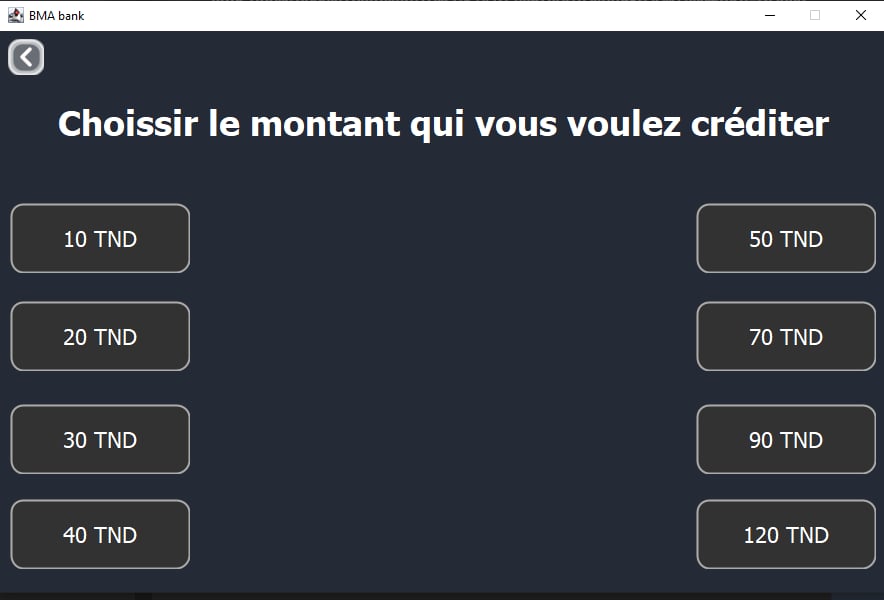
La principale fenêtre contient 3 boutons :

* **Solde** : Permet de consulter le solde.
* **Créditer** : Permet de créditer un montant.
* **Débiter** : Permet de débiter un montant.

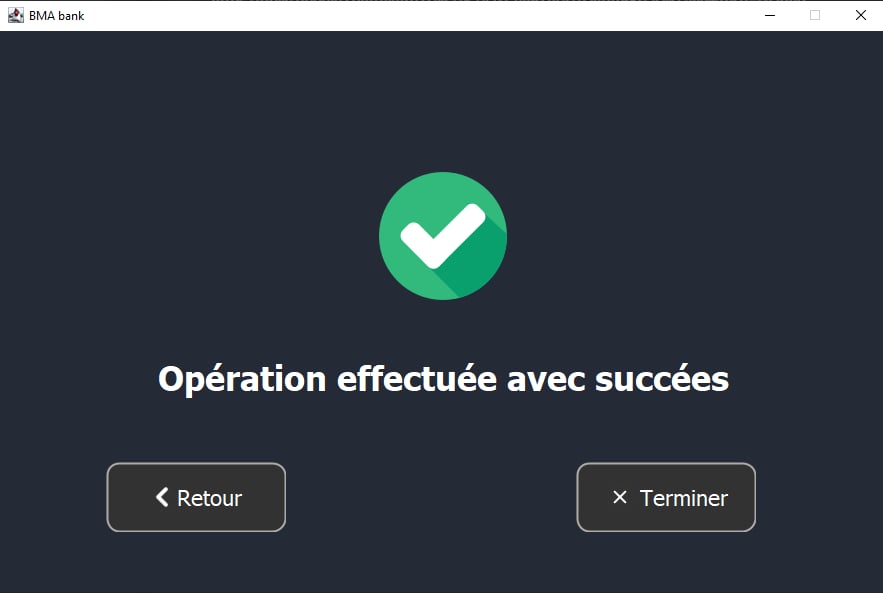
**Interface Solde:**

****

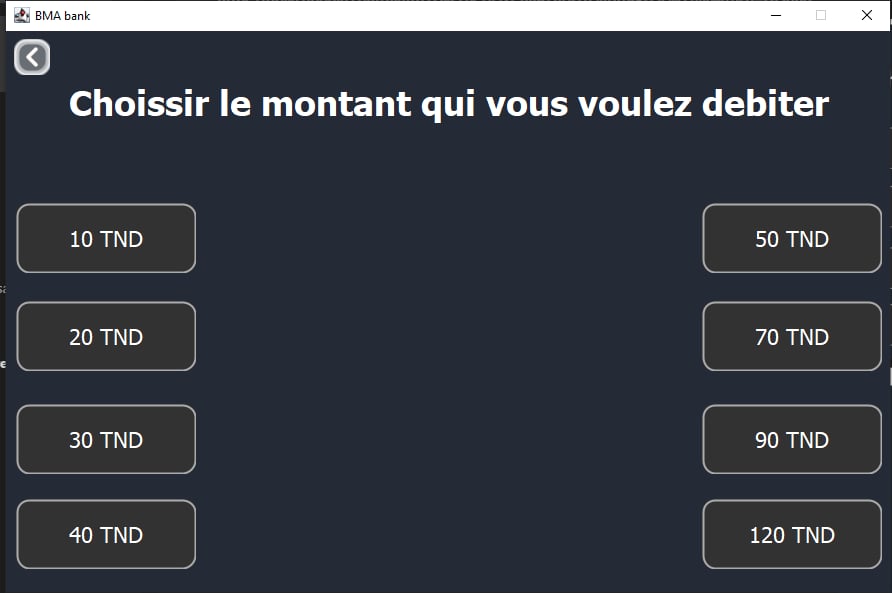
**Interface Créditer:**

****

Si le montant qui vous avez choisi est valide et votre balance ne dépasse pas le montant autorisé.

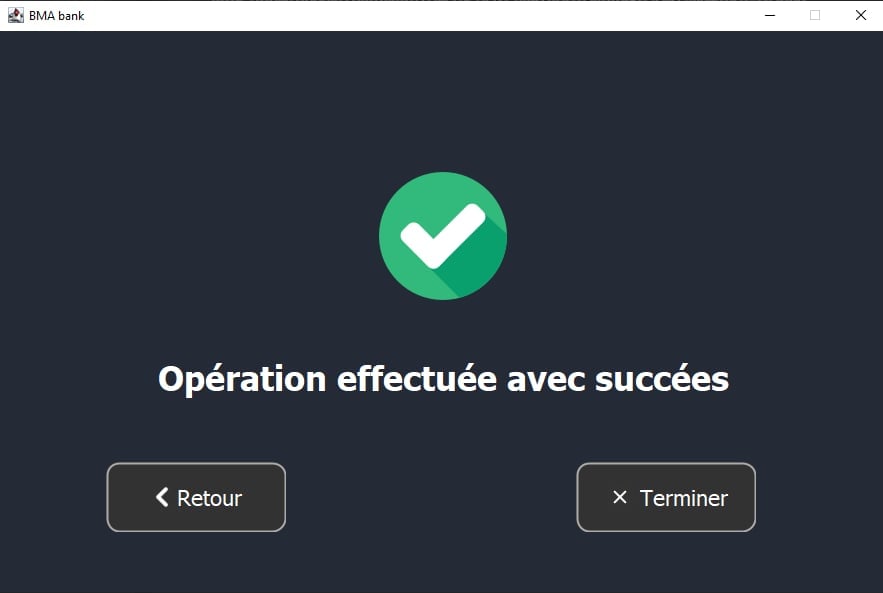


**Interface Créditer:**

****

**1ére Cas :**

Si le montant qui vous avez choisi est valide et votre solde est suffisant pour le retrait d’argent.



**2ére Cas :**

Si vous n’avez pas le solde suffisant pour le retrait de l’argent.

