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



**Niveau : ING1\_INF**

**Groupe : TP1**

**Réaliser par :**

**Haithem Ben Slama**

**Enseignant : Mr Ramzi Mahmoudi**



Table des matières

Table des matières iii

Chapitre 1. Environnement de développement JavaCard 4

I.1 Installation du JDK 4

I.1.1 Téléchargement et installation du JDK 18.0.2.1: 4

I.1.2 Ajouter le Variable d’environnement JAVA\_HOME : 5

I.2 Installation de l’IDE - Eclipse : 8

I.3 Test du l’IDE Eclipse L’indispensable « Hello Word » : 9

I.4 Installation du Java Card Development Kit 2 .2.2 : 13

I.4.1 Ajouter le Variable d’environnement JC\_HOME : 15

I.5 Mise à jour des plugins Eclipse-JCDE: 17

I.6 Conclusion : 19

Chapitre II. Développement d’une application coté Serveur 20

II.1 Création d’un nouveau Projet JavaCard : 20

II.2 Création d’une applet Javacard: 21

II.3 Codage de notre applet : 25

II.3.1 Enoncé : 25

II.3.2 Implémentation : 25

II.4 Outils de simulation : 28

II.4.1 JCWDE : simulateur sans conservation d'état : 28

II.4.2 CREF : simulateur avec conservation d'état : 32

Chapitre III. Programmation d’une application coté client 37

III.1 Création de l’application client sous Eclipse : 37

III.1.1 Création d’un nouveau projet Java : 37

III.1.2 Ajout de la librairie « apduio » dans le classpath : 38

III.1.3 Création de la classe principale : 39

III.1.4 Implémentation de l’application client : 42

III.1.5 Utilisation de l'application cliente avec un simulateur – JCWDE : 46

Chapitre IV. Réalisation du Mini-Projet 51

IV.1 Interface Graphique ( Partie Client ) : 51

IV.1.1 WindowBuilder Introduction : 51

IV.1.2 Installation de WindowBuilder : 52

IV.1.3 Comment utiliser WindowBuilder : 54

IV.1.4 Les interfaces de notre application Client : 57

IV.1.5 Implémentation des méthodes nécessaires : 64

IV.2 Partie Serveur (Back-End) : 66

IV.2.1 Déclaration des variables et des constantes : 66

IV.2.2 Implémentations des méthodes nécessaires : 67

Chapitre 1. Environnement de développement JavaCard

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. Installation du JDK

Il ne faut pas confondre le JDK (Java Development Kit) avec le JRE (Java Runtime Environment) :

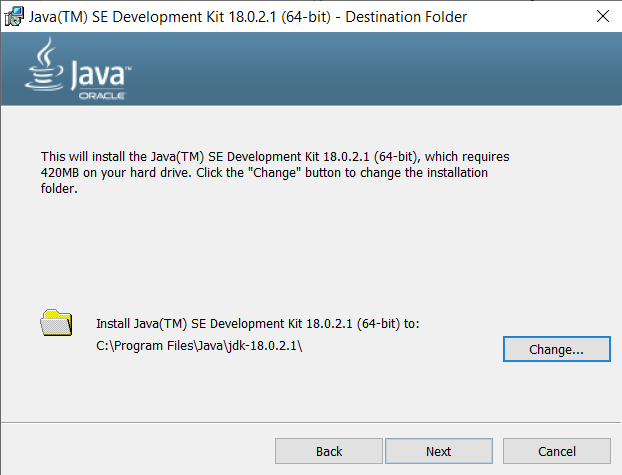
* La JRE ne contient que les outils nécessaires pour exécuter des applications Java ;
* Le JDK permet également d’exécuter des applications Java mais il permet en plus de compiler du code source java pour en faire des applications compréhensibles par la JRE, le JRE est inclut dans le JDK.
  + 1. Téléchargement et installation du JDK 18.0.2.1:

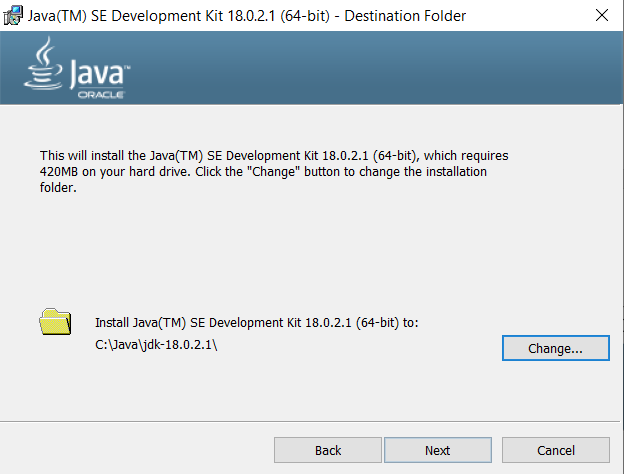
1. Télécharger et installer l’exécutable **jdk-18.0.2.1\_windows-x64\_bin.exe** à partir du lien suivant :

<https://download.oracle.com/java/18/archive/jdk-18.0.2.1_windows-x64_bin.exe>

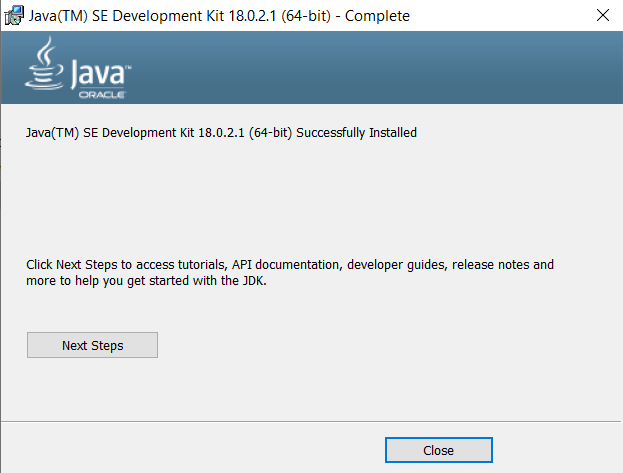
1. Une fois l’exécutable est téléchargé ouvrer le et cliquer sur Next.



1. Modifier le chemin d’installation avec **C:\Java\jdk-18.0.2.1\**

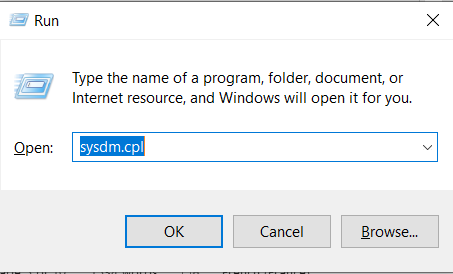


1. Finalement le JDK est installer avec succès

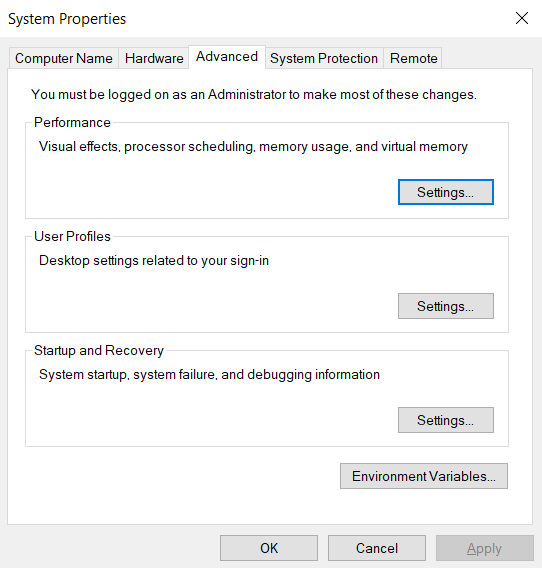


* + 1. Ajouter le Variable d’environnement JAVA\_HOME :

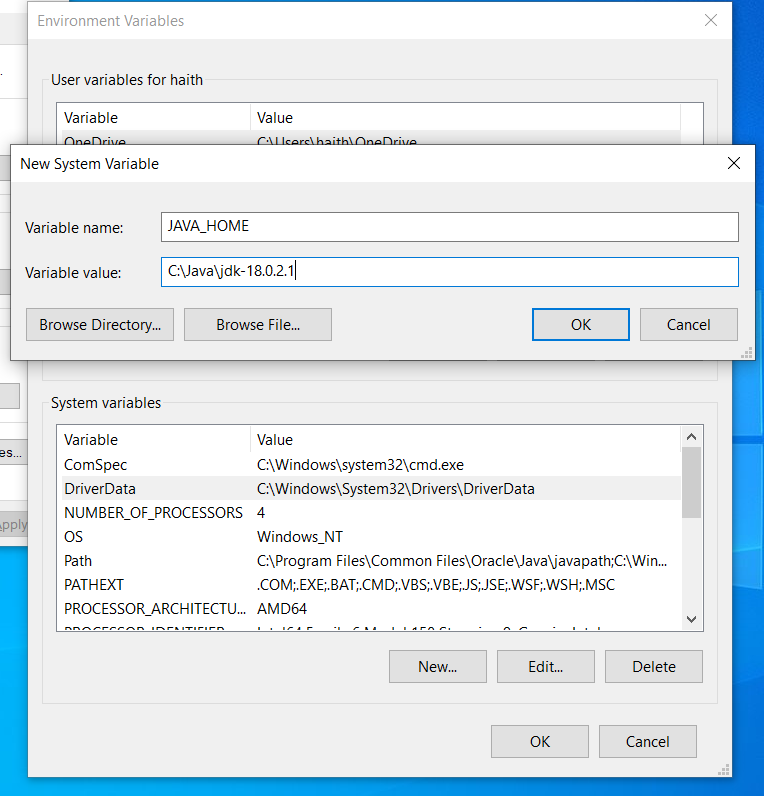
1. Cliquer sur Microsoft Windows icon PNG PIC | PNG Mart + R puis tapez **sysdm.cpl**



1. Cliquer sur **Advanced** puis **Environment Variables**



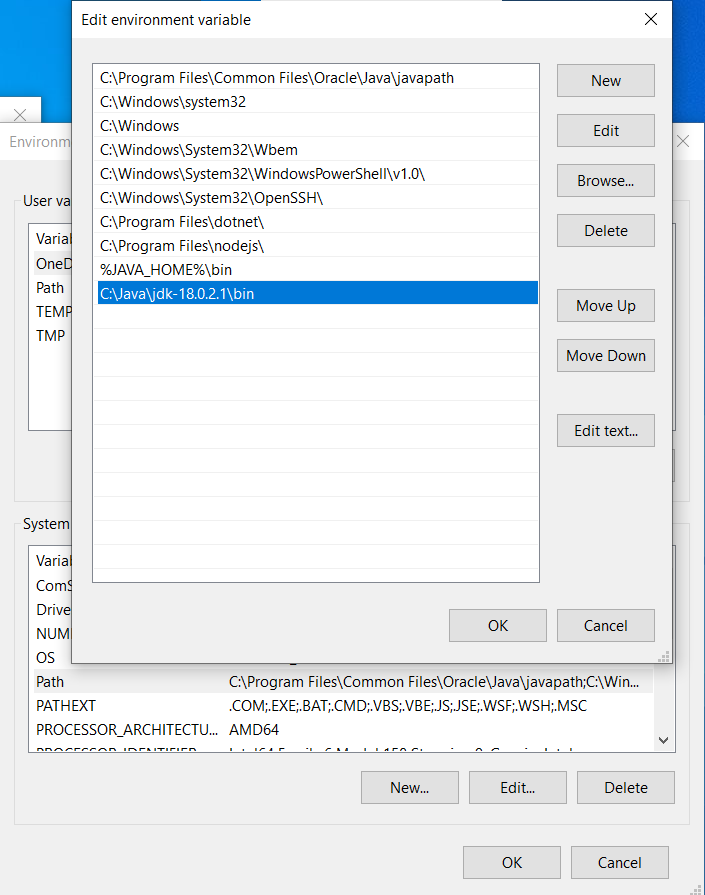
1. Créer une nouvelle variable d’environnement (coté système) nommée **JAVA\_HOME** contenant le chemin **C:\Java\jdk-18.0.2.1\**



1

2

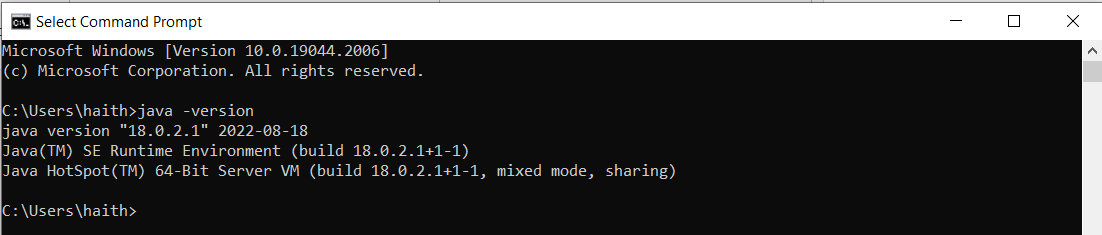
1. Modifier la variable Path (coté système) en ajoutant vers sa fin : **%JAVA\_HOME%\bin** et **C:\Java\jdk-18.0.2.1\bin**



Double clic sur Path

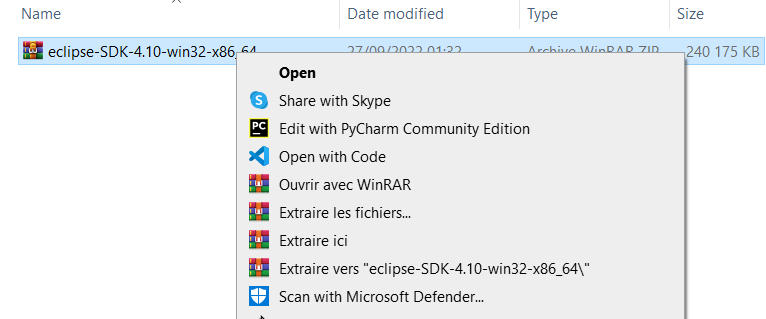
1. Pour vérifier le bon fonctionnement de votre JDK, il suffit de taper sous la console DOS la commande :

java –version

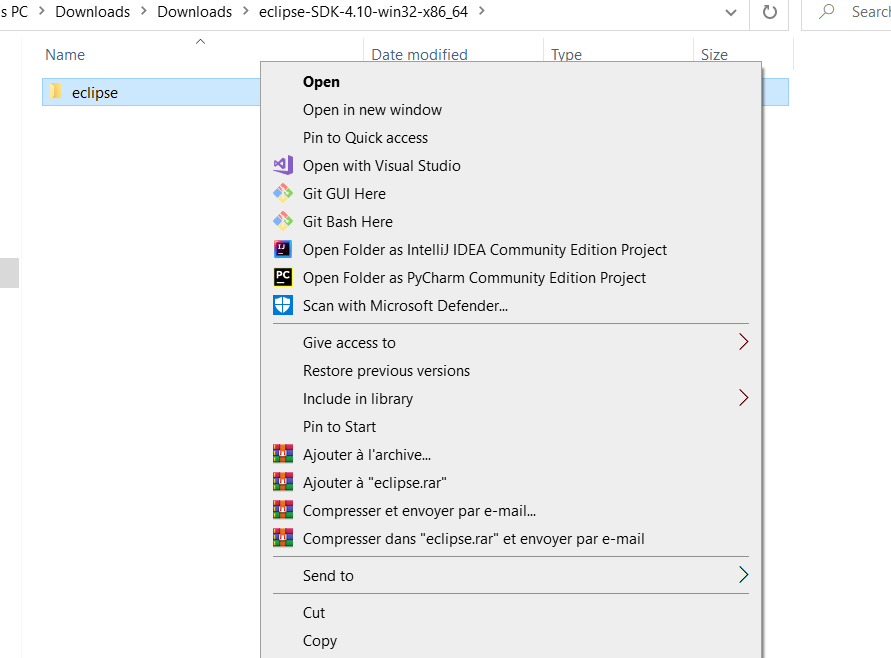


* 1. Installation de l’IDE - Eclipse :

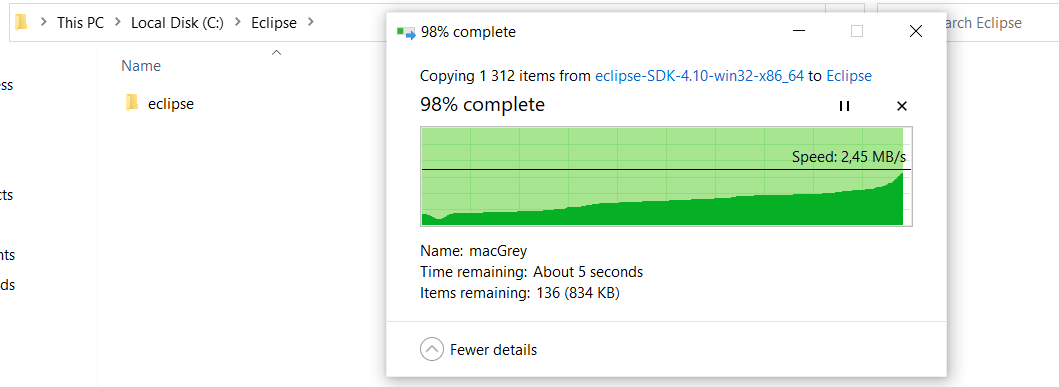
1. Télécharger le zip **eclipse-SDK-4.10-win32-x86\_64.zip** à partir du lien suivant : <http://www.eclipse.org/downloads/download.php?file=/eclipse/downloads/drops4/R-4.10-201812060815/eclipse-SDK-4.10-win32-x86_64.zip>
2. Une fois le zip est téléchargé Extraire le.



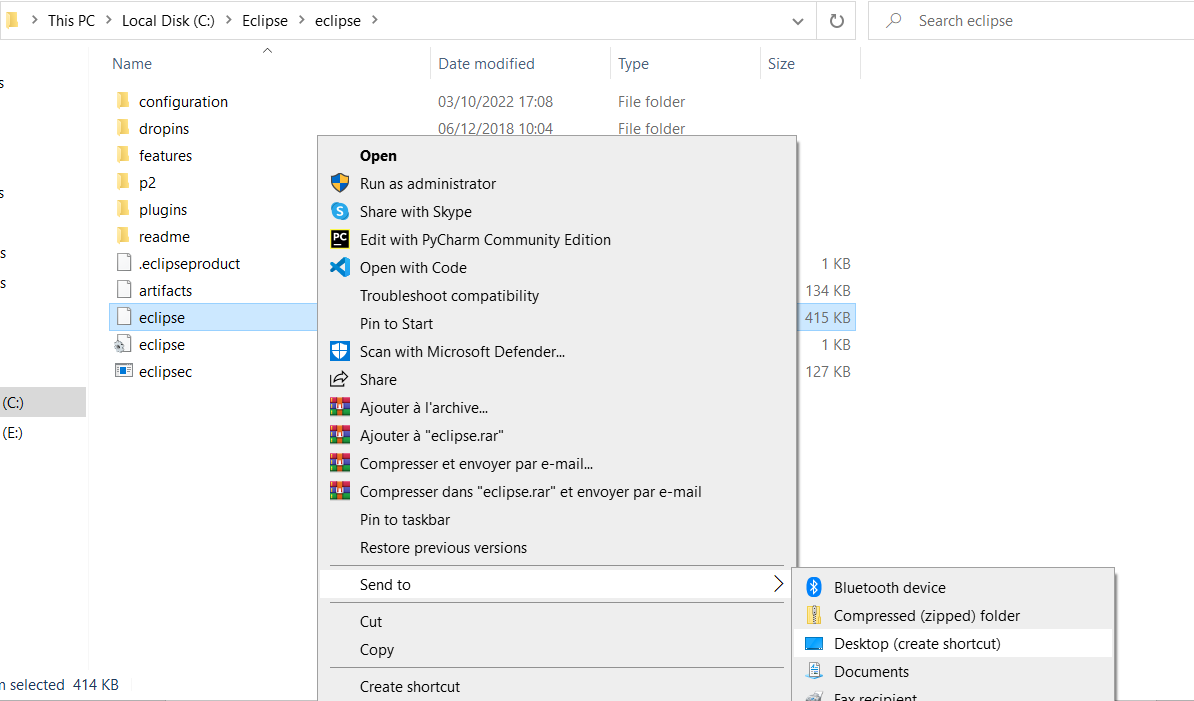
1. Après l’extraction copier le contenu fi fichier **eclipse-SDK-4.10-win32-x86\_64**



1. Aller sur **C :** et créer un nouveau répertoire nommé **Eclipse** puis coller le dossier copier précédemment.



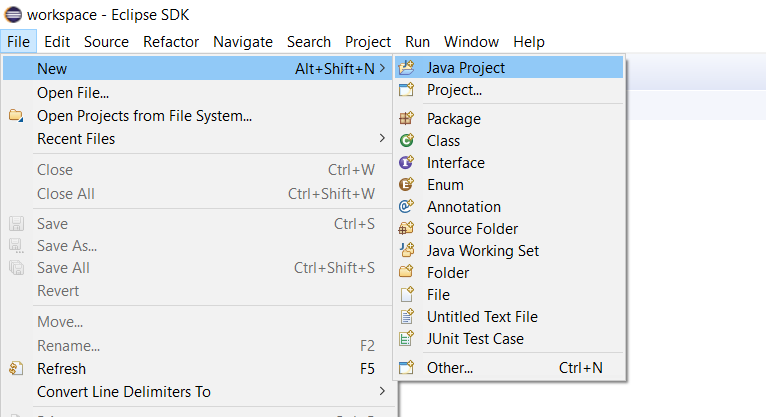
1. Pour créer un raccourci Eclipse sur bureau. Il suffit d’aller **C:\Eclipse\eclipse**, clique droite sur **eclipse** puis **send to > desktop (create shortcut)**



* 1. Test du l’IDE Eclipse L’indispensable « Hello Word » :

Comme initiation, nous allons créer une petite application Java avec l’IDE Eclipse qui va afficher à l'écran "Hello, World !» :

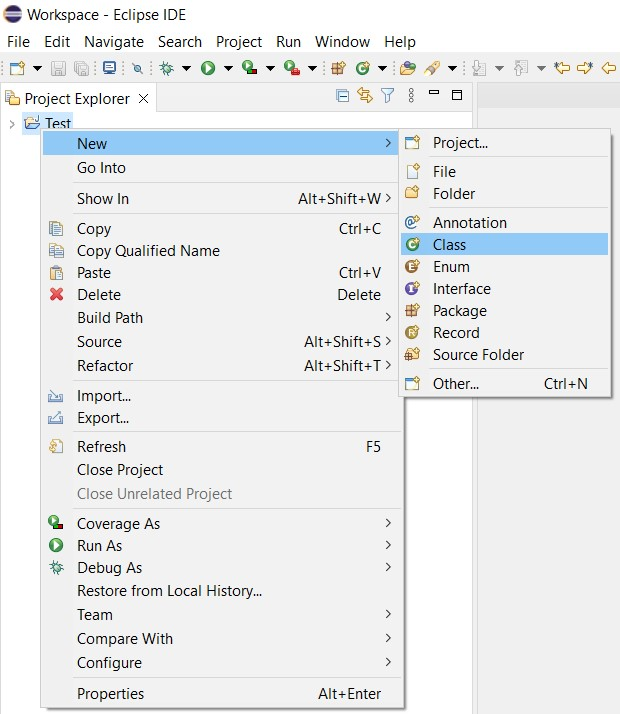
1. Cliquez sur **File > New > Java Project**



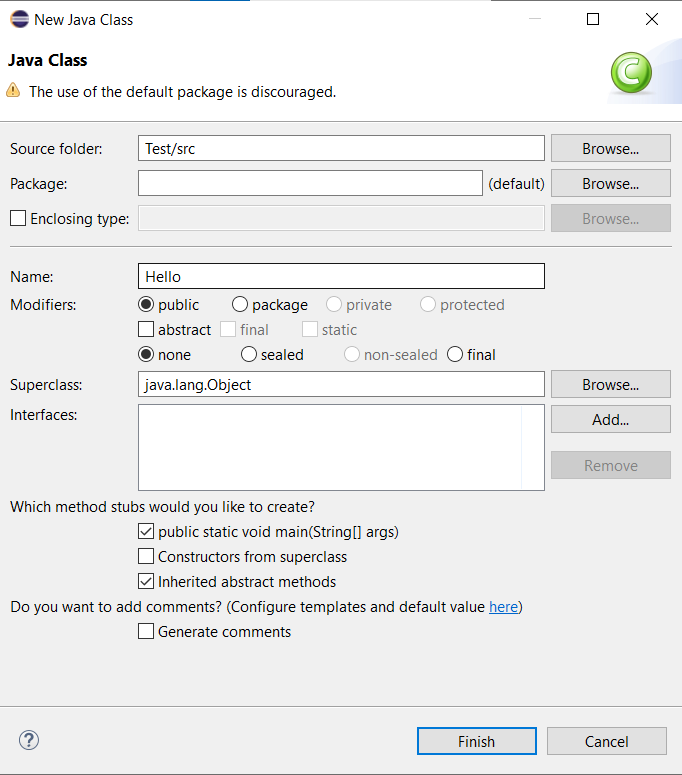
1. Nommez le projet puis cliquez sur **Finish**



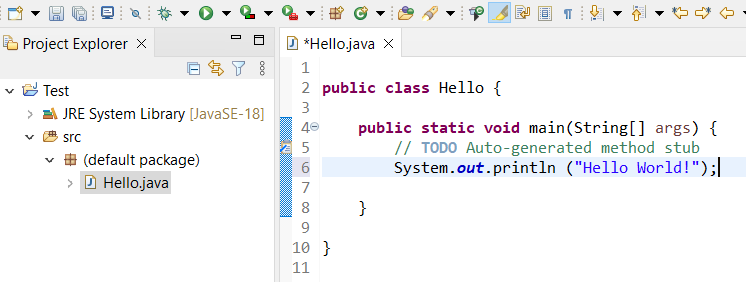
1. Cliquer avec le bouton droit sur le projet hello, on sélectionne New -> Class :



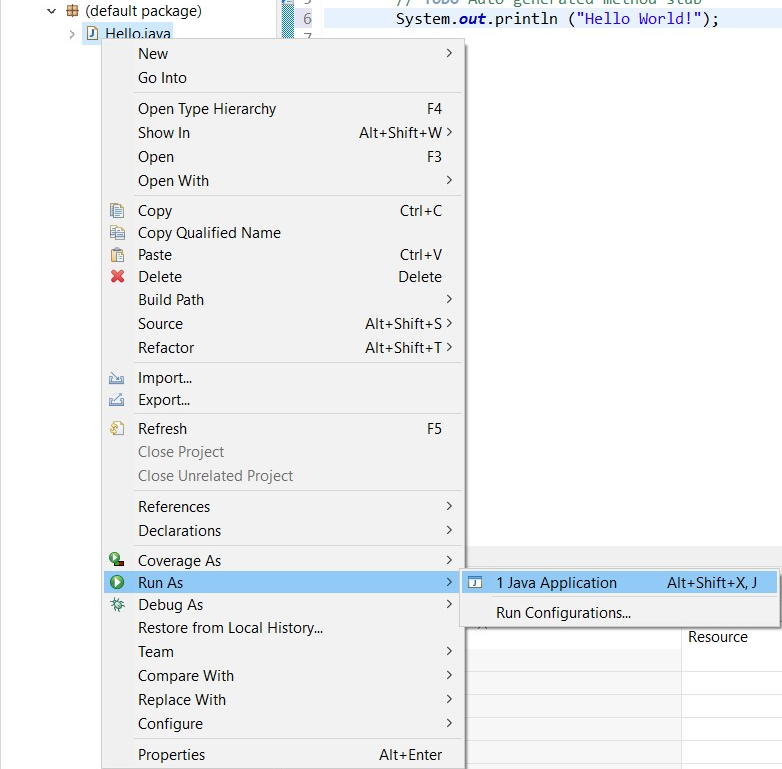
1. Dans la fenêtre qui s'ouvre, on indique le nom de la classe, **Hello**, et **on coche la case** indiquant que l'on **souhaite qu'elle contienne une méthode main :**



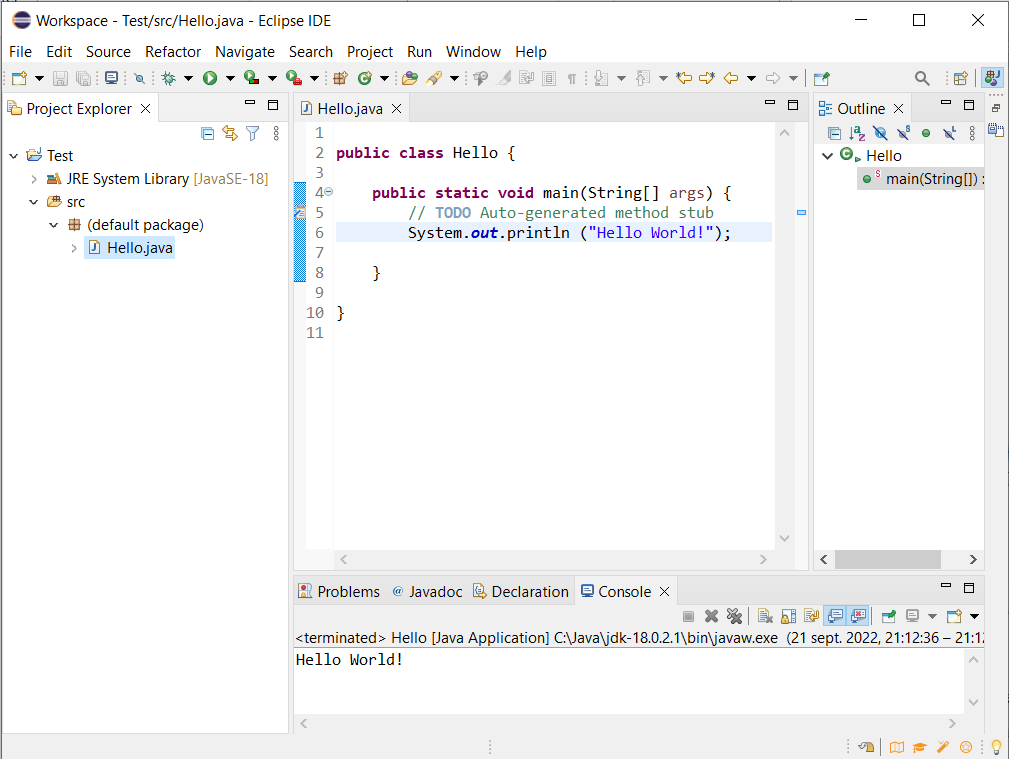
1. Cliquer alors sur le bouton **Finish**, Il ne reste qu'à compléter la méthode main, en ajoutant **System.out.println ("Hello World!");**



1. Pour tester notre programme, **cliquez droit** sur la classe **Hello.java**, puis **Run As** > **Java Application**



1. L'affichage de l'exécution s'effectue sous l'onglet "**console**" en bas de la fenêtre principale :



* 1. Installation du Java Card Development Kit 2 .2.2 :

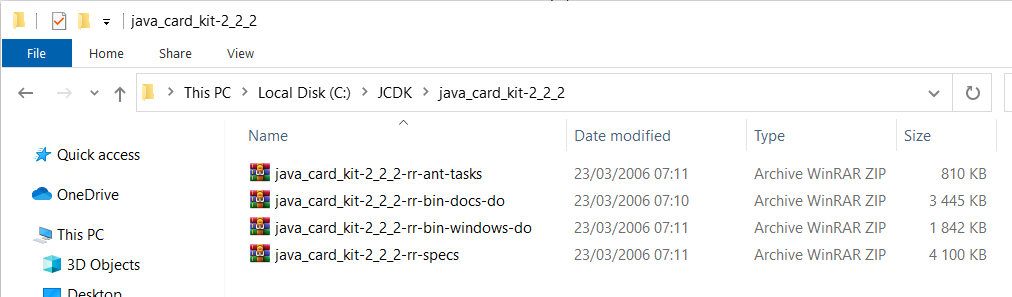
C’est un ensemble des méthodologies, des outils et des concepts informels imposés par les normes adaptées à l'environnement d'utilisation afin de favoriser la production et la maintenance de composants logiciels de qualité.

Le génie logiciel est généralement réservé aux logiciels complexes de grande envergure et non aux applications ou programmes simples. Le développement n'est toutefois qu'une phase du processus. Les ingénieurs logiciels sont responsables de la conception des systèmes, alors que les programmeurs sont chargés du codage permettant leur implémentation.

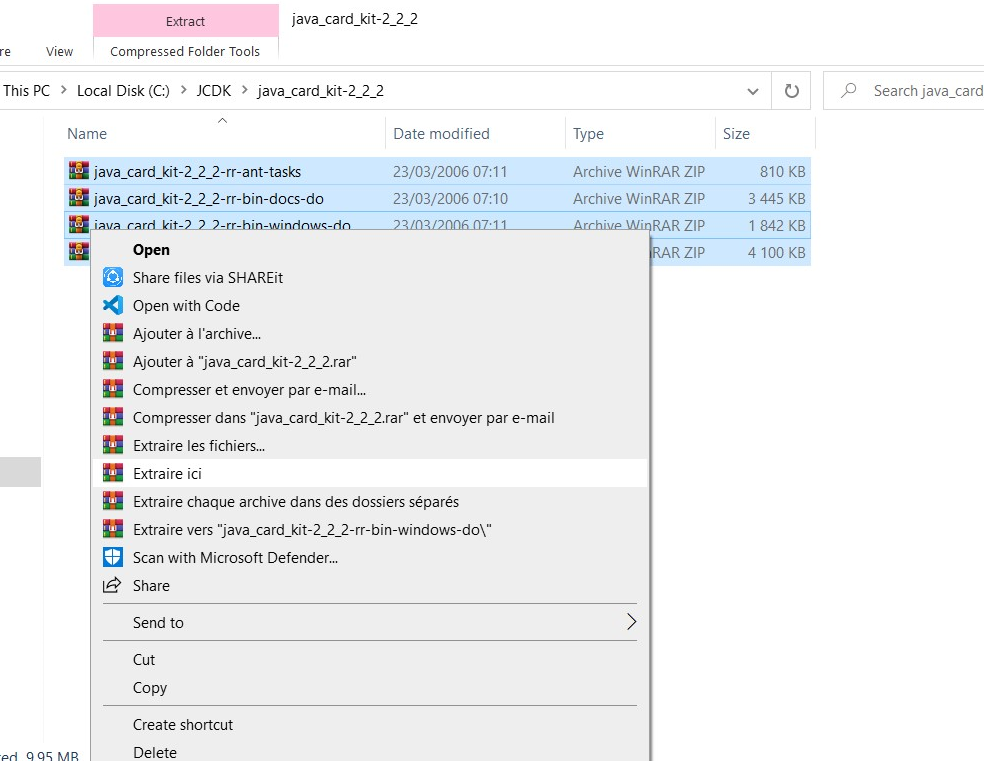
1. Télécharger le zip **java\_card\_kit-2\_2\_2-windows** à partir du lien suivant :

<https://download.oracle.com/otn-pub/java/java_card_kit/2.2.2/java_card_kit-2_2_2-windows.zip>

1. Créer un nouveau répertoire **C:\JCDK**
2. Décompresser l’archive **java\_card\_kit-2\_2\_2-windows.zip** après l’avoir téléchargé dans le répertoire **C:\JCDK**.
3. Aller dans le répertoire **C:\JCDK\java\_card\_kit-2\_2\_2**, de cette manière vous devriez voir apparaître les fichiers suivants :



1. Décompresser tous les fichiers du répertoire courant **C:\JCDK \java\_card\_kit-2\_2\_2 :**

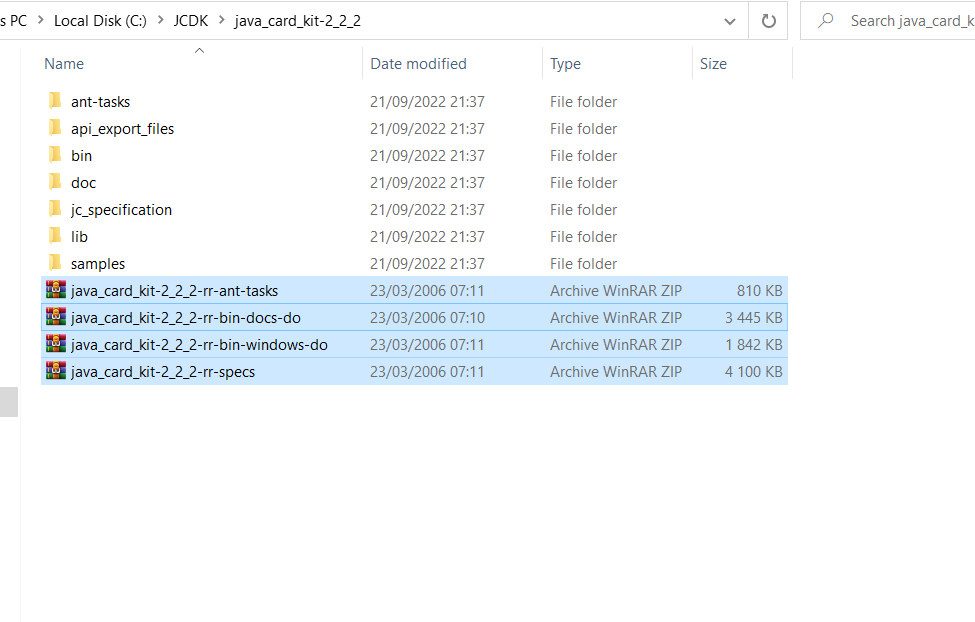


* De cette manière vous devriez voir apparaître les fichiers suivants :

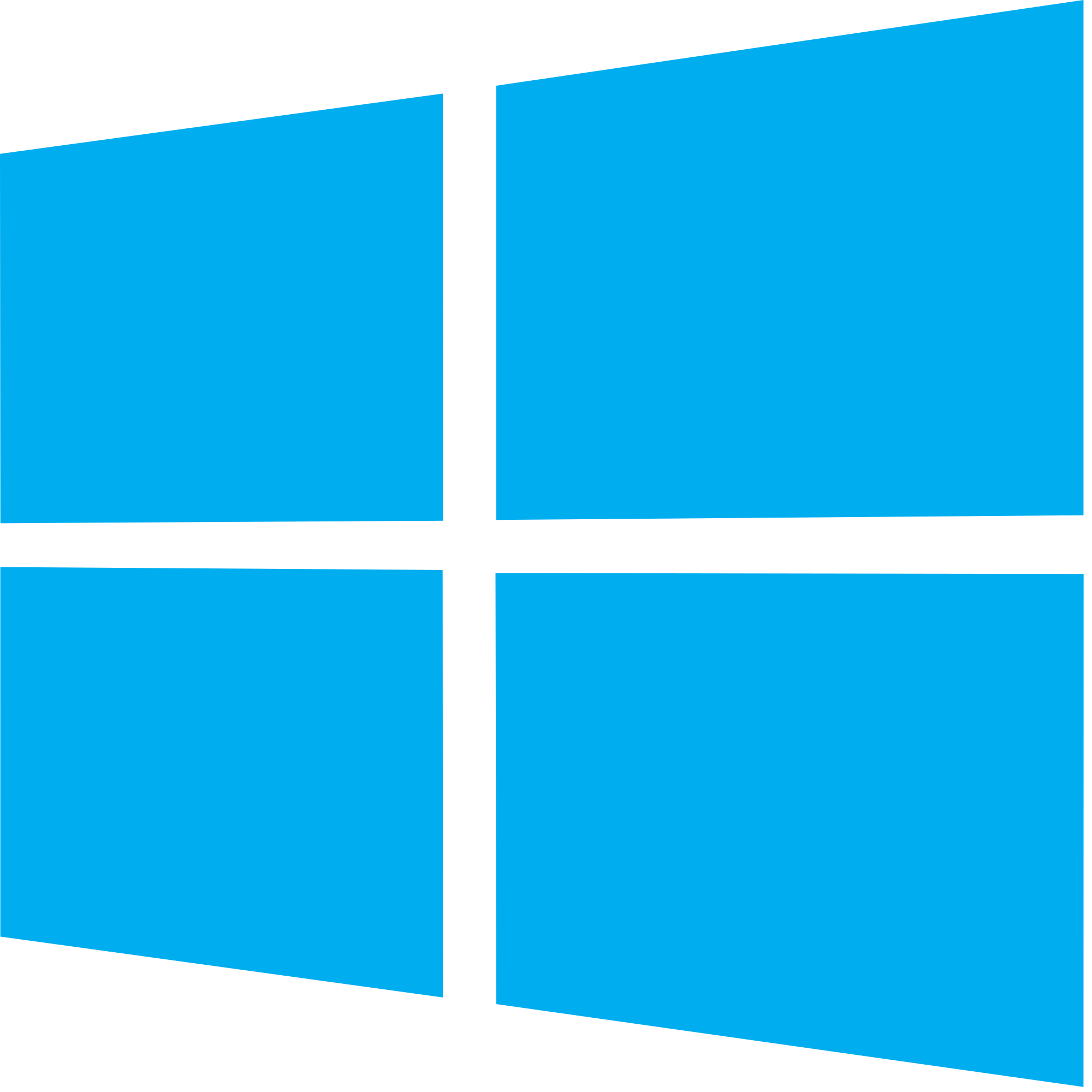
- **bin** : contient les outils du JCDK (simulateur, ...)

- **lib** : contient les librairies du JCDK

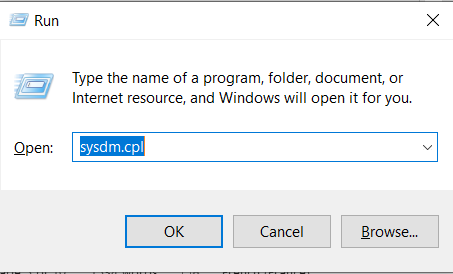
- **jc\_specifications\specs\api\html** : contient l'API du JCDK (très utile)



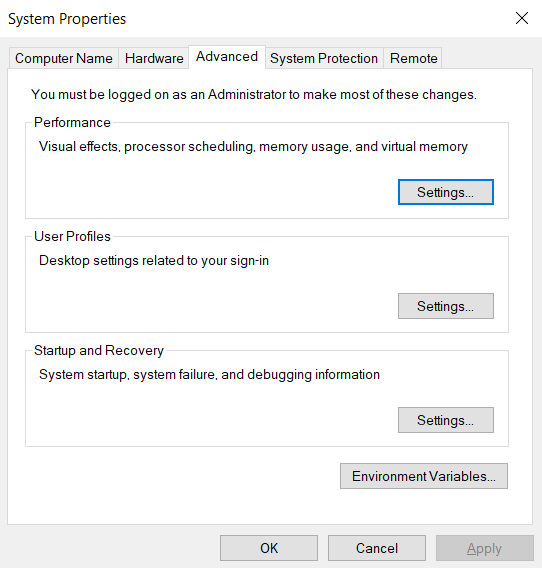
* + 1. Ajouter le Variable d’environnement JC\_HOME :



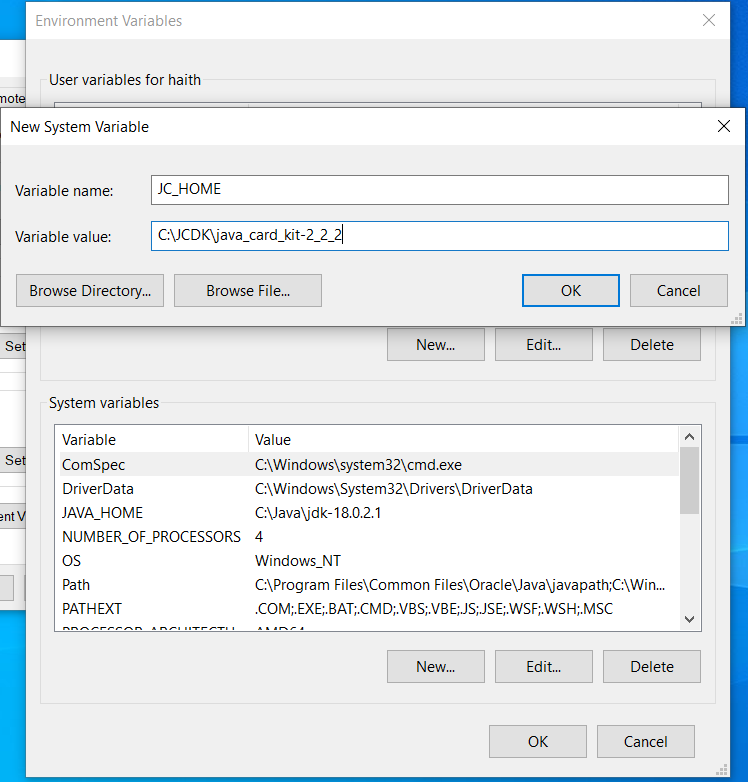
1. Cliquer sur + R puis tapez **sysdm.cpl**



1. Cliquer sur **Advanced** puis **Environment Variables**



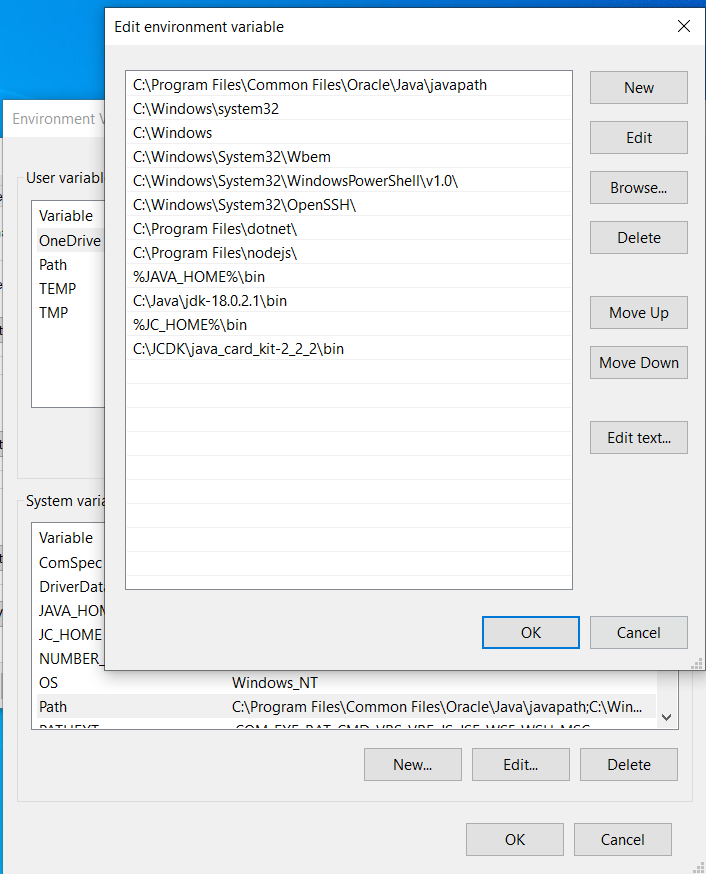
1. Créer une nouvelle variable d’environnement (coté système) nommée **JC\_HOME** contenant le chemin **C:\JCDK\java\_card\_kit-2\_2\_2**



1

2

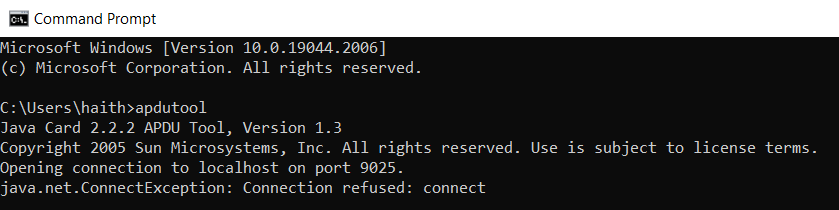
1. Modifier la variable Path (coté système) en ajoutant vers sa fin : **JC\_HOME%\bin** et **C:\JCDK\java\_card\_kit-2\_2\_2\bin**



Double clic sur Path

1. Vérifiez la réussite de configuration en tapant dans une console de commandes :

apdutool



* 1. Mise à jour des plugins Eclipse-JCDE:

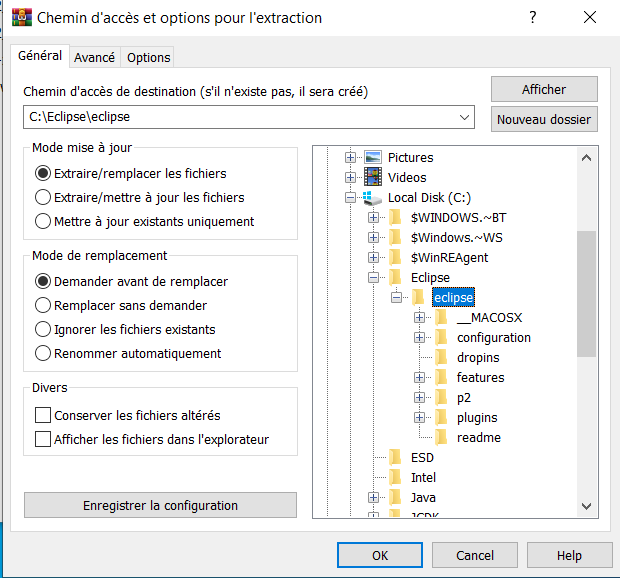
C’est un ensemble des méthodologies, des outils et des concepts informels imposés par les normes adaptées à l'environnement d'utilisation afin de favoriser la production et la maintenance de composants logiciels de qualité.

Le génie logiciel est généralement réservé aux logiciels complexes de grande envergure et non aux applications ou programmes simples. Le développement n'est toutefois qu'une phase du processus. Les ingénieurs logiciels sont responsables de la conception des systèmes, alors que les programmeurs sont chargés du codage permettant leur implémentation.

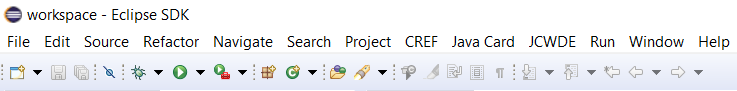
1. Télécharger le zip **eclipse-jcde-0.2.zip** à partir du lien suivant :

<https://sourceforge.net/projects/eclipse-jcde/files/eclipse-jcde/eclipse-jcde-0.2.zip/download>

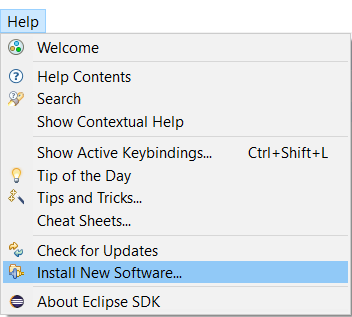
1. Décompresser l’archive **eclipse-jcde-0.2.zip** après l’avoir téléchargé dans le répertoire **C:\Eclipse\eclipse\**



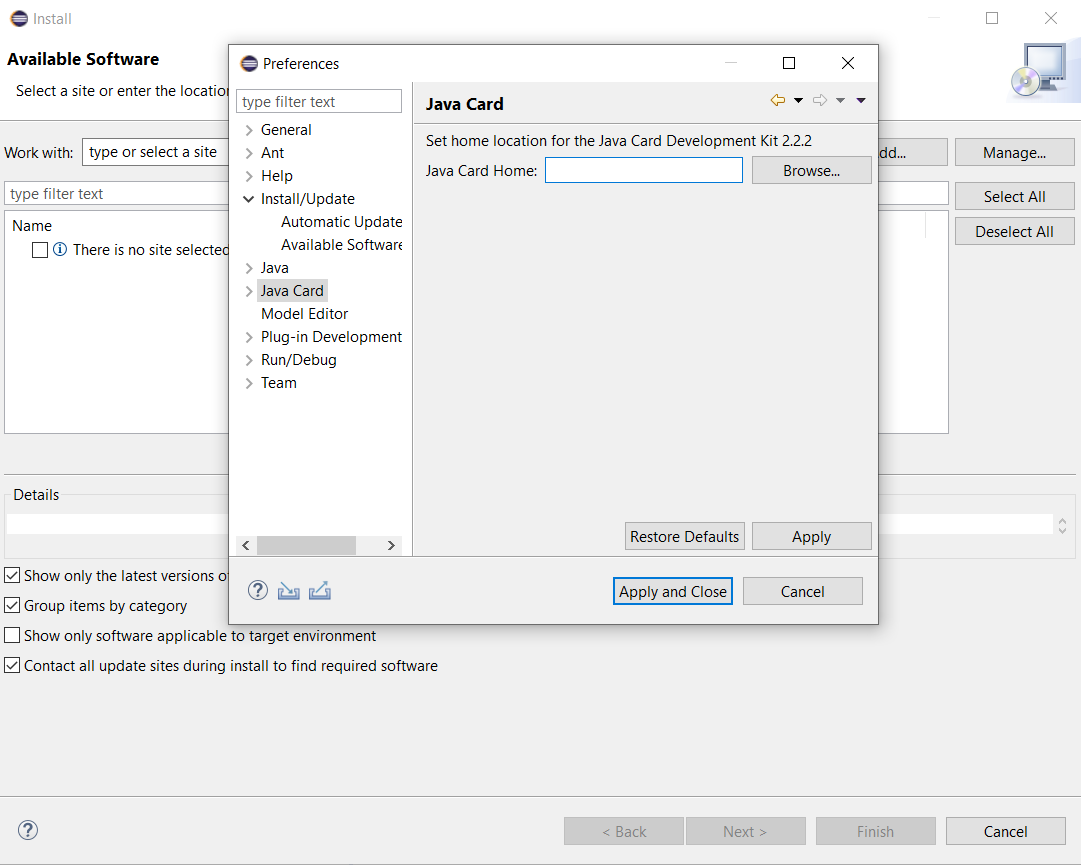
1. Ouvrer Eclipse, vous remarquez que les plugins sont ajoutés dans la barre de menu.



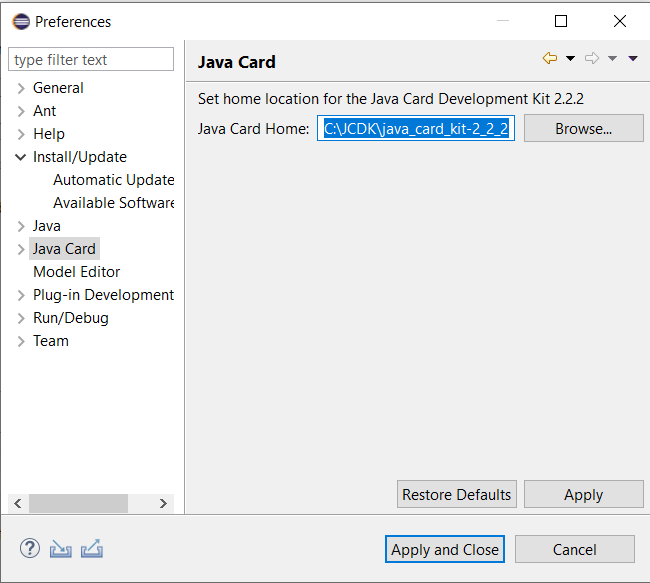
1. Maintenant allez sur **Help>Install New Software…**



1. Cliquez sur **Manage** puis **JavaCard.**



1. Ajouter le chemin de Java Card Home **C:\JCDK\java\_card\_kit-2\_2\_2** puis cliquer sur **Apply and Close.**



* 1. 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

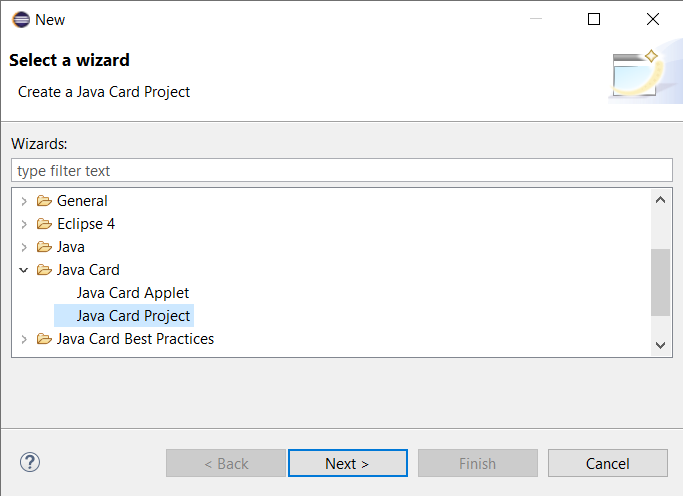
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. Création d’un nouveau Projet JavaCard :

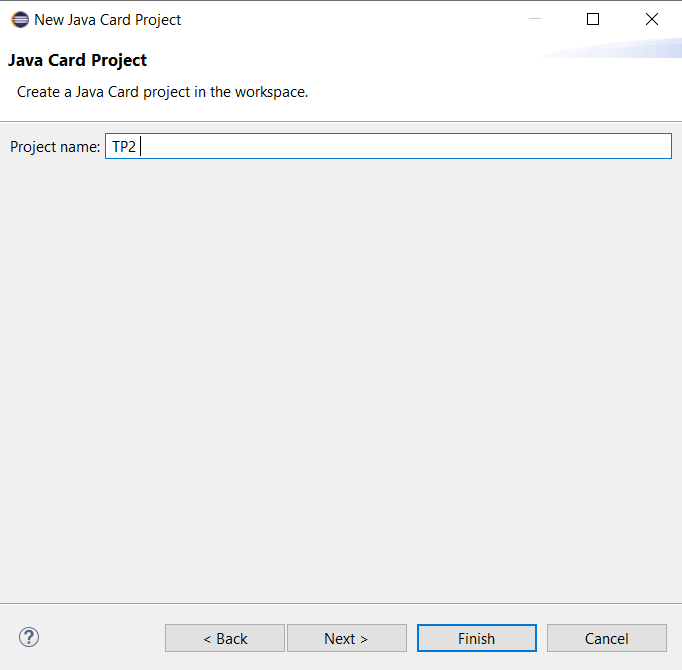
1. Ouvrir Eclipse, Dans la barre du Menu cliquer sur **File>New Project>Other**



1. Une fenêtre sera affichée, sélectionner **Java Card** puis **Java Card Project** et cliquer sur **Next**:

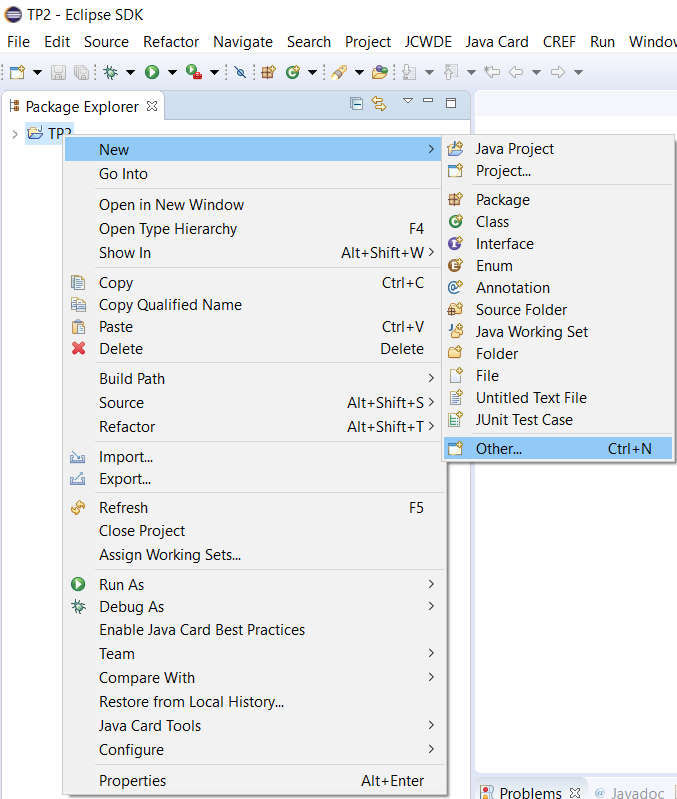


1. Donner alors un nom au projet puis cliquer sur Finish

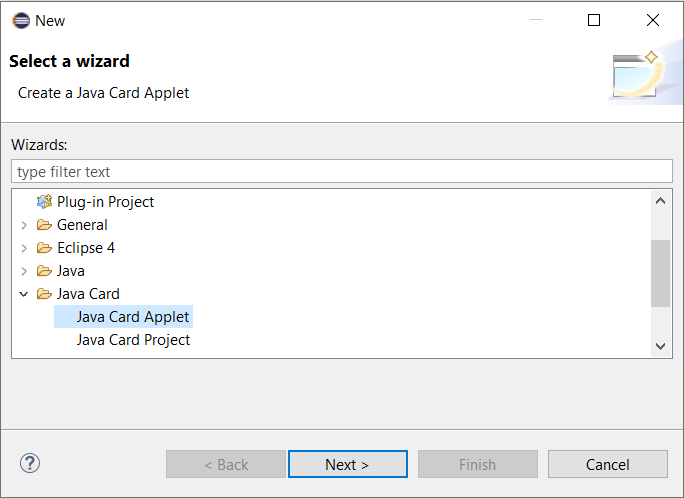


* 1. Création d’une applet Javacard:

1. A ce niveau le projet est créé avec succès. Click droite sur le Projet qui apparait dans le Package Explorer puis **New>Other**

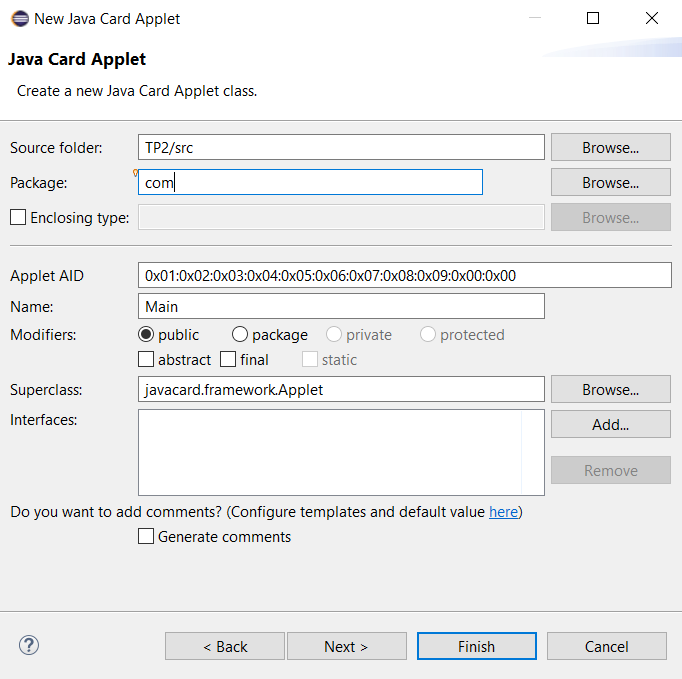


1. Sélectionner **JavaCard Applet** puis **Next :**



**Dans la technologie Java Card, chaque applet est identifiée et sélectionnée par un identificateur (AID).**

1. L’AID sera généré automatiquement. Il suffit de choisir alors le nom de l’applet dans le champ **Name**, et **le nom du Package**. Nous les appellerons respectivement "com" et "Main"



1. Cliquer alors sur le bouton **Finish**. Eclipse vient de générer automatiquement le squelette de notre Applet :

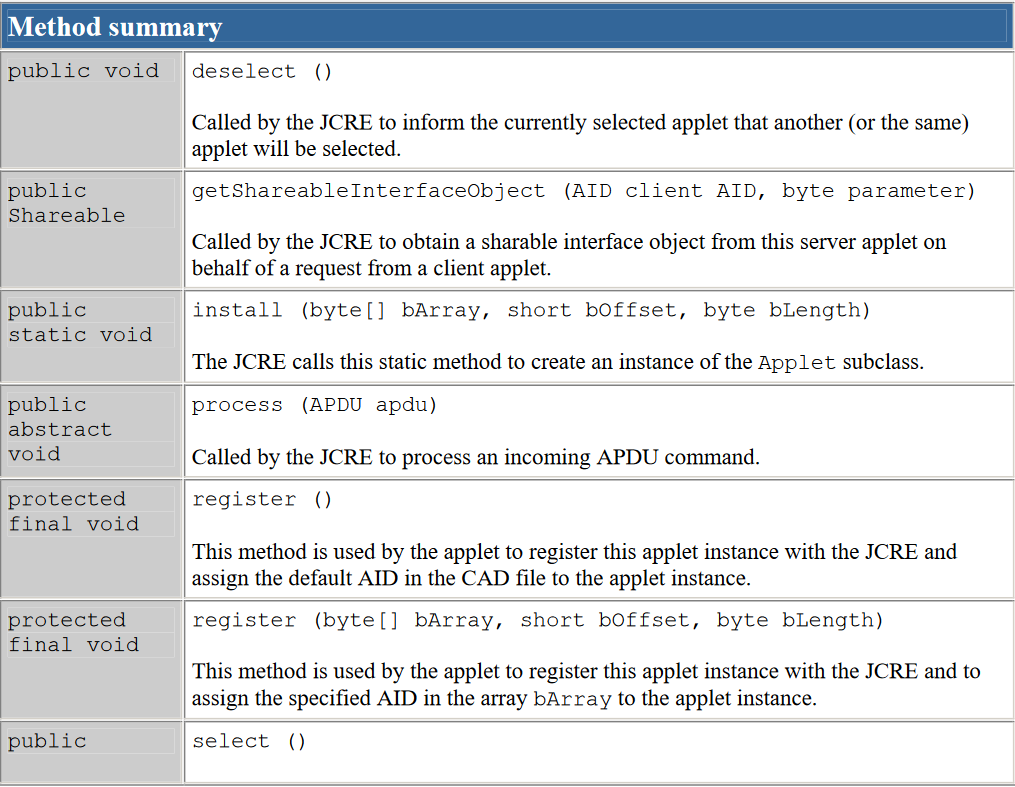


**Applet**

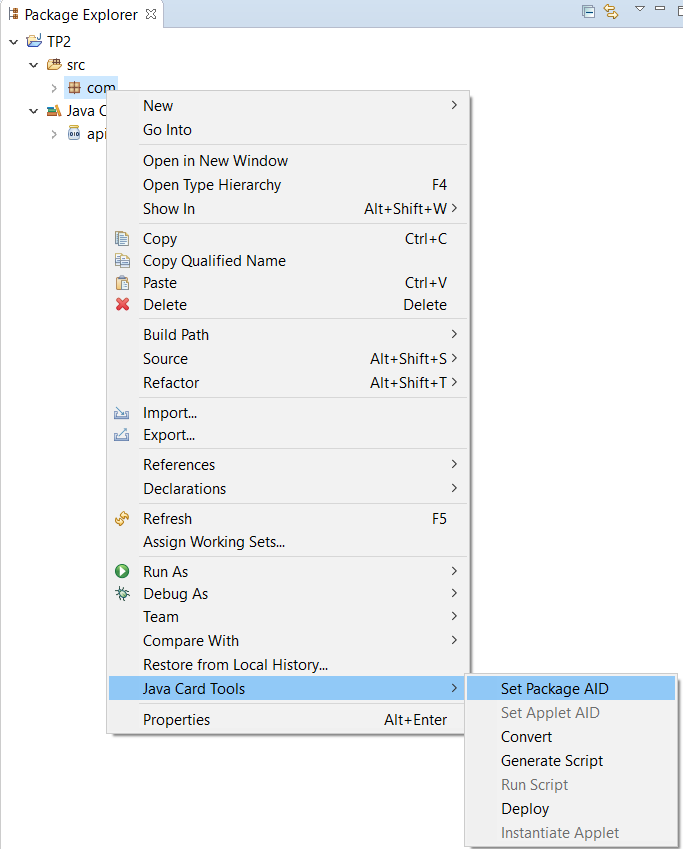
**Package**

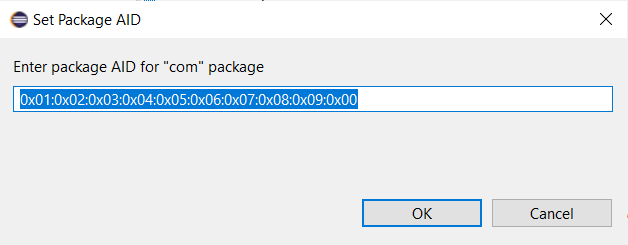
**Squelette de notre Applet**

**Remarque :**Les méthodes et objets clés de l'applet :



1. On doit définir l’AID du package. Alors clic droit sur le package **com** > **Java Card Tools > Set Package AID** :





À chaque paquetage Java est assigné un AID. Cette  
convention de nommage est conforme à la spécification de la carte à puce définie dans  
l’ISO 7816.

* 1. Codage de notre applet :
     1. Enoncé :

Nous allons créer une applet compteur. L'applet comportera 4 fonctions :  
 - incrémenter le compteur  
 - décrémenter le compteur  
 - interroger le compteur  
 - initialiser le compteur à une valeur donnée

* + 1. Implémentation :
* **Etape 1. Ajouter API JavaCard :**

L’API Java Card consiste en un ensemble de classes spécialisées dans la  
programmation d’applications de smart card conformément à la norme ISO 7816. Elle  
contient un package principal javacard.framework pour programmer une applet carte  
qui définit :

* La classe **javacard.framework.Applet** superclasse des applets résident sur la  
  carte. Une applet Java Card doit étendre de cette classe.
* La classe **javacard.framework.APDU** encapsule les commandes APDU.
* L’interface **javacard.framework.ISO7816** définit un ensemble de mots d’état  
  qui signalent les erreurs courantes des applets.
* La classe **javacard.framework.ISOException** pour gérer les exceptions

**package** com;

**import** javacard.framework.APDU;

**import** javacard.framework.Applet;

**import** javacard.framework.ISO7816;

**import** javacard.framework.ISOException;

* **Etape 2. Déclarer les attributs et les constantes :**

Pour cela, nous allons définir :  
 **Constantes** : type primitif **byte**

* **CLA\_MONAPPLET** identifie la famille de commandes APDU supportées par notre applet JavaCard.
* Les codes **INS** définissent le type d’instruction des commandes APDU traitées par notre applet.  
  **Attributs :** type primitif **byte**
* **Compteur** définit la variable à incrémenter ou décrémenter.

**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;

* **Etape 3. Définition des méthodes publiques qu'elle doit obligatoirement implémenter :**

On définit tout d'abord :

1. **la méthode install () : création et enregistrement de l'objet Applet :** 
   * Notre applet doit implémenter la méthode **install**() pour créer une instance d’applet et doit enregistrer l’instance au sein du JCRE en invoquant la méthode **register().**
   * La méthode **install()** prend un vecteur d’octets comme paramètre. Ce vecteur contient les paramètres d’installation pour l’initialisation et la personnalisation de l’instance d’applet.

Constructeur de notre Applet

**private** MonApplet() {

compteur=0;

}

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

**new** MonApplet().register();

}

* **Une fois installée**, l’applet Java Card reste **inactive** jusqu’à ce qu’elle soit explicitement **sélectionnée**. La classe javacard.framework.Applet fournit une implémentation par défaut pour les méthodes **select()** et **deselect().**

1. **la méthode process () : Traitement des commandes APDU**
   * Une fois l’applet sélectionnée, le JCRE fait suivre toutes les commandes APDU (y compris la commande SELECT) à la méthode process() de l’applet.
   * Dans la méthode process(), l’applet interprète chaque commande APDU et exécute la tâche spécifiée par la commande. Pour chaque commande APDU, l’applet répond au CAD en envoyant une réponse APDU qui informe le CAD du résultat du traitement de la commande APDU. La méthode process() de la classe javacard.framework.Applet est une méthode de type abstract: une sous-classe de la classe Applet doit redéfinir cette méthode pour implémenter les fonctions de l’applet.

**public** **void** process(APDU apdu) **throws** ISOException {

//Extraire le buffer APDU

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

//Vérifier si c'est un APDU de selection ou non

**if**(**this**.selectingApplet()) **return**;

**if**(buffer[ISO7816.***OFFSET\_CLA***]!= ***CLA\_MONAPPLET***){

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

}

//Choisir l'instruction qui sera exécutée a partir du INS

**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**;

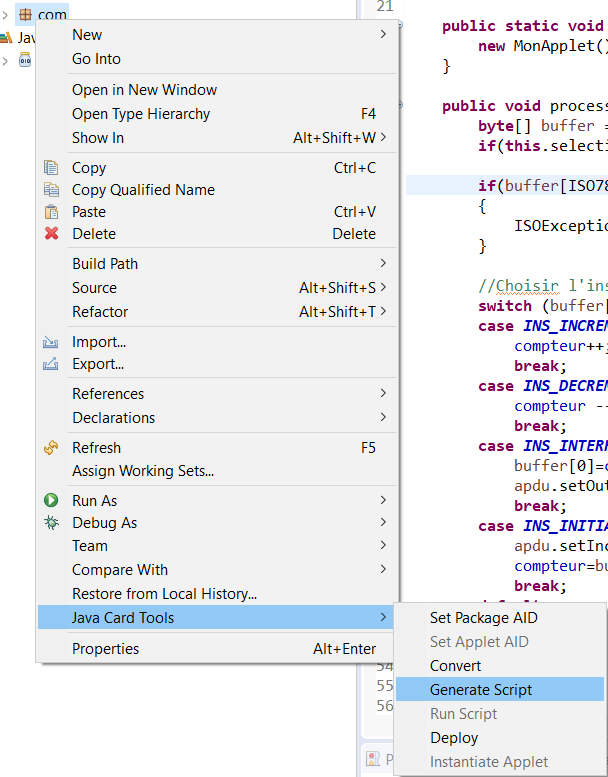
**default**:

}

}

* 1. Outils de simulation :

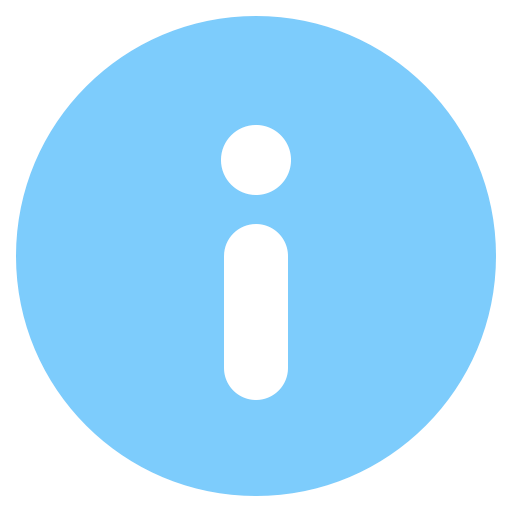
Avant de commencer les simulations, faire un clic droit sur le package **com**, sélectionner **Java Card Tools** puis **Generate Script**.



Ceci a pour conséquence de générer automatiquement les APDU nécessaires à l'upload, l'instanciation (installation) et la sélection de l'applet sur une Javacard. **com.javacard** contient alors 3 scripts :  
 ✓ **cap-download.script** : upload de l'applet  
 ✓ **create-MonApplet.script** : instanciation (installation) de l'applet  
 ✓ **select-MonApplet.script** : sélection de l'applet

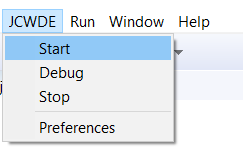
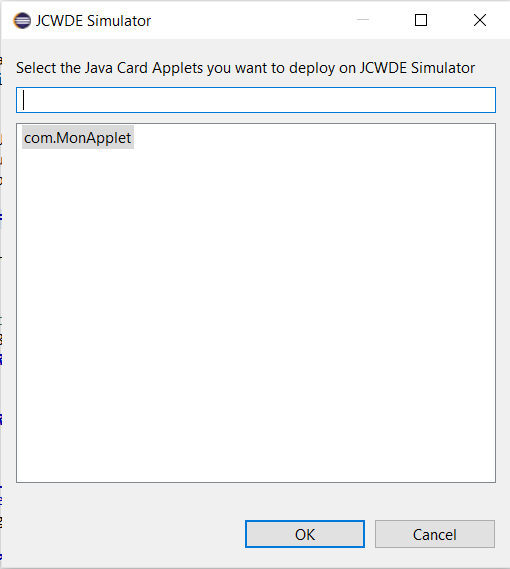
**APDUTOOL : envoi/réception d'APDU**

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.



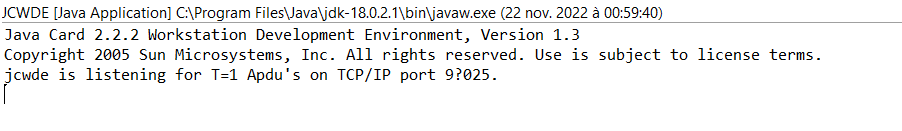
* + 1. JCWDE : simulateur sans conservation d'état :

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

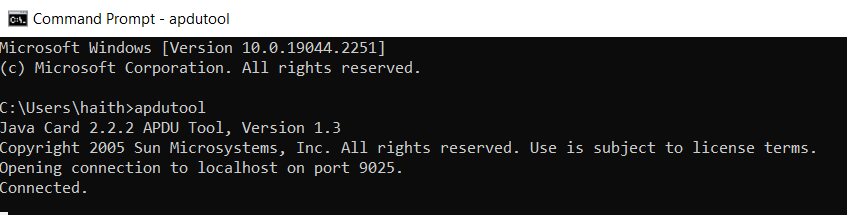
****

**Remarque :**

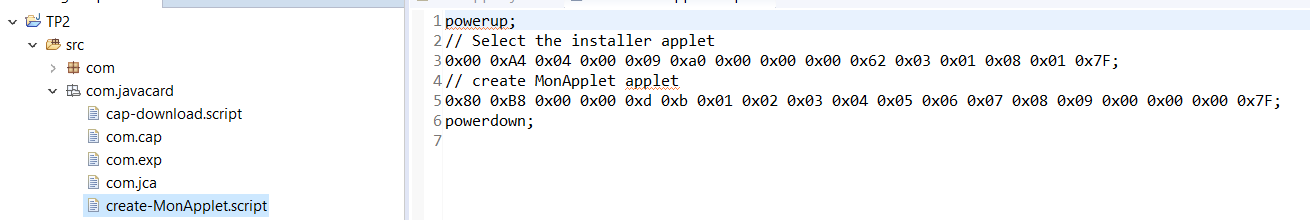
Dans la console un message s’affiche comme suit :

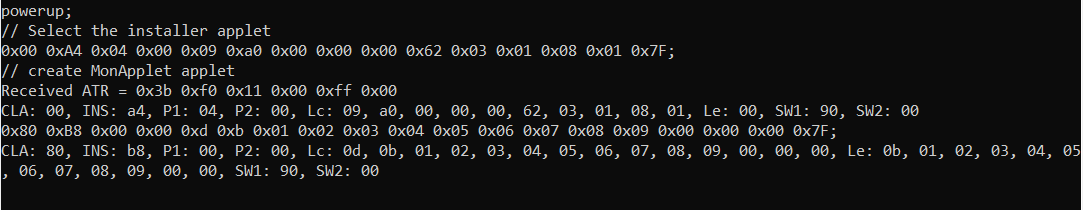


1. Cliquer sur le bouton **OK** pour lancer le simulateur JCWDE avec notre applet. Le simulateur est alors lancé et attend une connexion. Nous pouvons alors lancer l'outil **APDUTOOL**. Pour cela, ouvrir une invite de commandes, puis taper **apdutool** et valider :



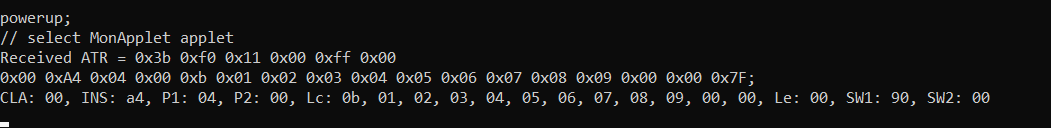
1. Installons notre applet en recopiant les APDU contenus dans le script create-MonApplet.script (sauf la dernière instruction **powerdown**) :





* Nous voyons que la carte répond " positivement " avec un status word 0x9000.

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



* De nouveau, la carte répond " **positivement** " avec un status word **0x9000.**

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

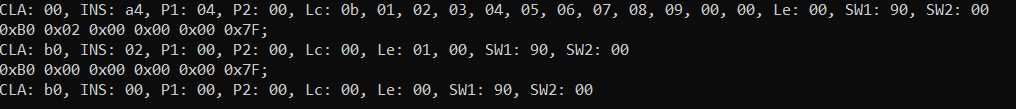
0xB0 0x02 0x00 0x00 0x00 0x7F;



Compteur=0

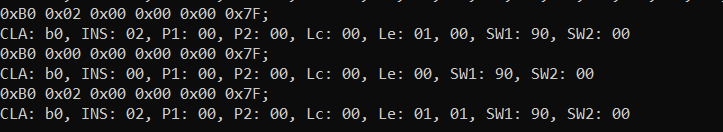
1. Nous allons l'incrémenter le compteur (**INS = 0x00**) en envoyant l'APDU suivante :

0xB0 0x00 0x00 0x00 0x00 0x7F;



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

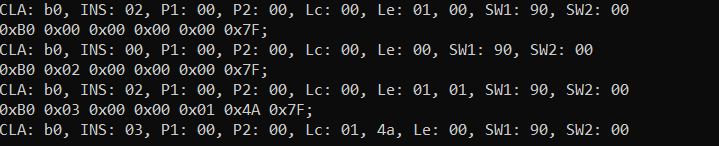
0xB0 0x02 0x00 0x00 0x00 0x7F;



Le compteur est désormais à 1

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

0xB0 0x03 0x00 0x00 0x01 0x4A 0x7F;

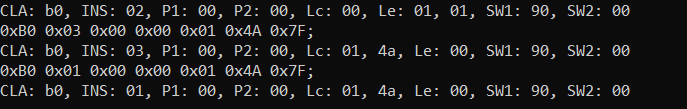


Le compteur est désormais à **4a**

La carte répond " positivement " avec un status word **0x9000**.

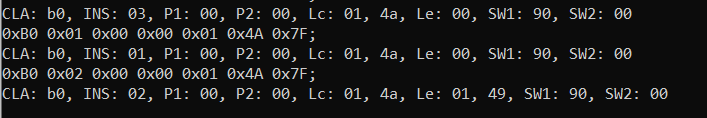
8. Décrémentons le compteur (**INS = 0x01**) en envoyant l'APDU suivante :

0xB0 0x01 0x00 0x00 0x01 0x4A 0x7F;



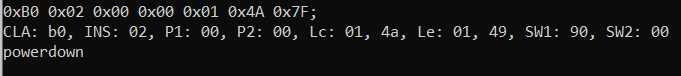
* Nous voyons que la carte répond " **positivement** " avec un status word **0x9000**

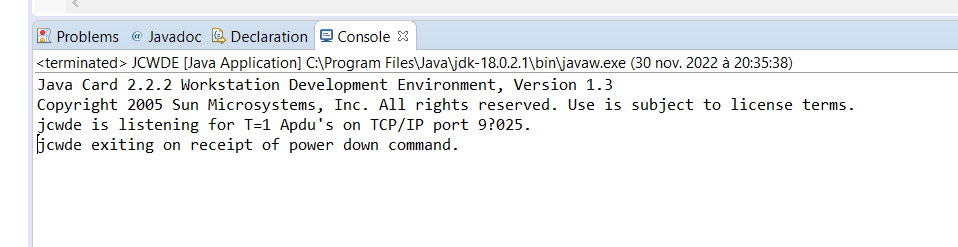
1. Interrogeons de nouveau le compteur (**INS = 0x02**) en envoyant l'APDU suivante :

  
  
  
  
 11. Enfin, Nous allons nous déconnecter du simulateur en tapant "**powerdown**;" dans **apdutool**, ce qui provoque la fermeture de **JCWDE** dans la console **d'Eclipse**

0xB0 0x02 0x00 0x00 0x01 0x4A 0x7F;

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



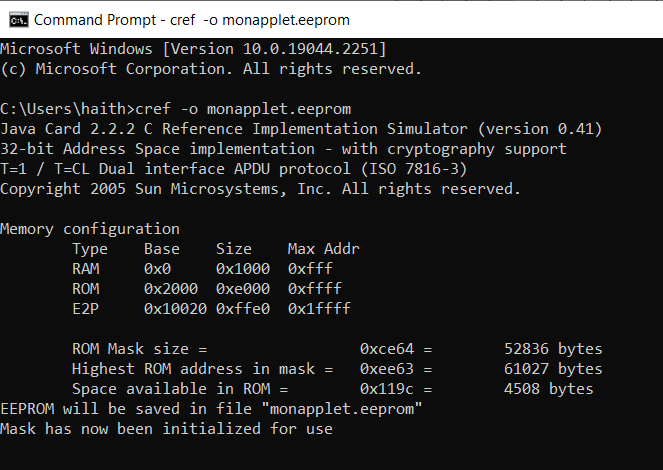


* + 1. CREF : simulateur avec conservation d'état :

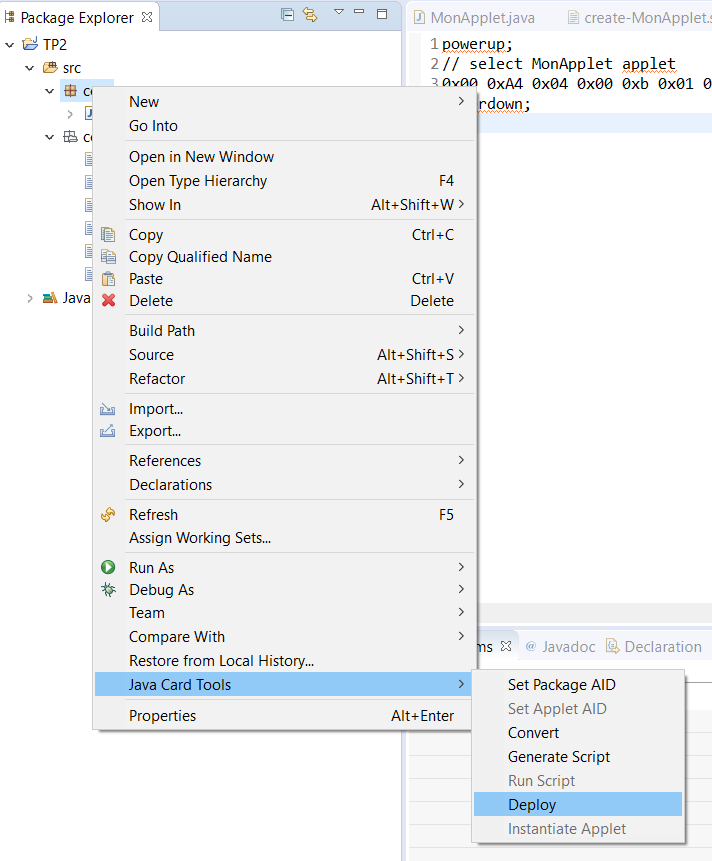
JCWDE a l'avantage de ne pas nécessiter l'upload de l'applet sur le simulateur, ce qui le rend plus souple par rapport à CREF. En revanche dès que JCWDE est stoppé (à la reception d'une commande **powerdown** par exemple), l'état de la carte n'est pas conservé (**exemple** : la valeur du compteur dans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

* Les options de ligne de commande que nous allons utiliser :  
  • **-i fichier\_eeprom :** initialise l'état du carte avec l'image **fichier\_eeprom**• **-o fichier\_eeprom :** enregistre l'état de la carte dans **fichier\_eeprom**

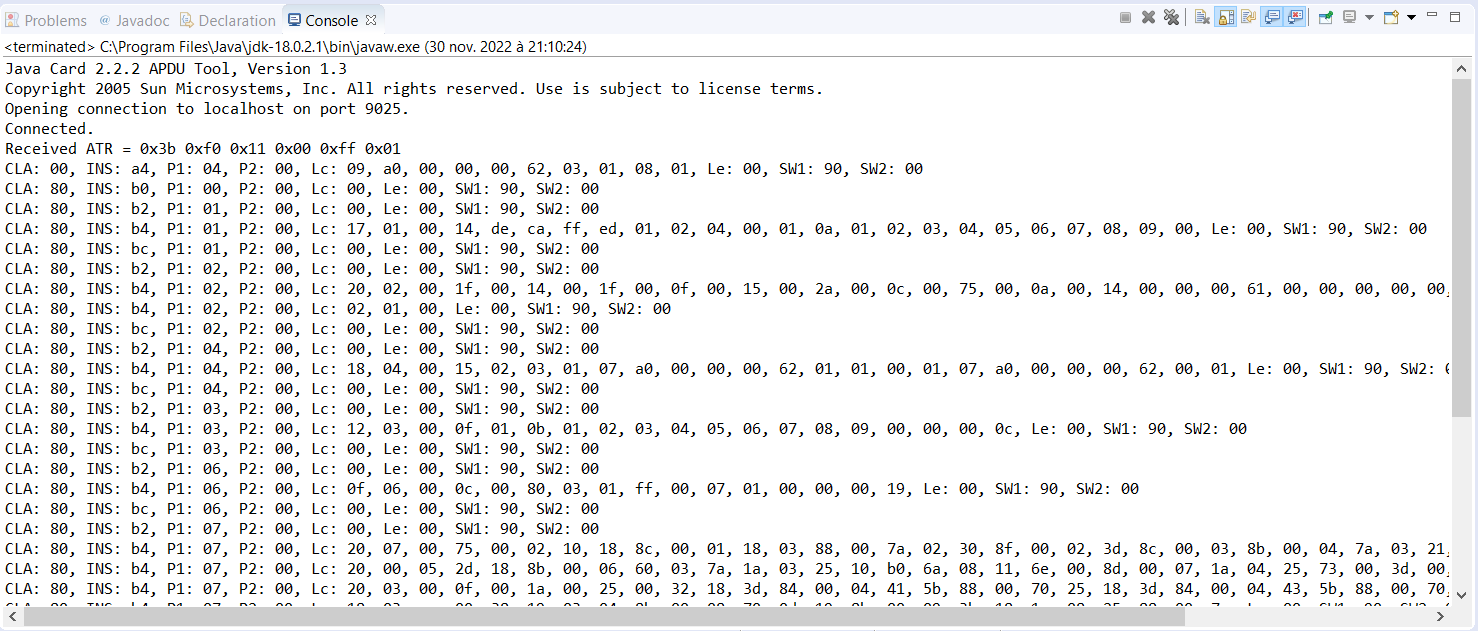
1. Nous pouvons lancer **CREF** et créer notre fichier image. Pour cela, ouvrir une invite de commandes, puis taper " **cref -o monapplet.eeprom** " et valider :



1. Uploadons maintenant notre applet. Pour cela, clic droit sur le package **com**>**JavaCard Tools**>**Deploy** :

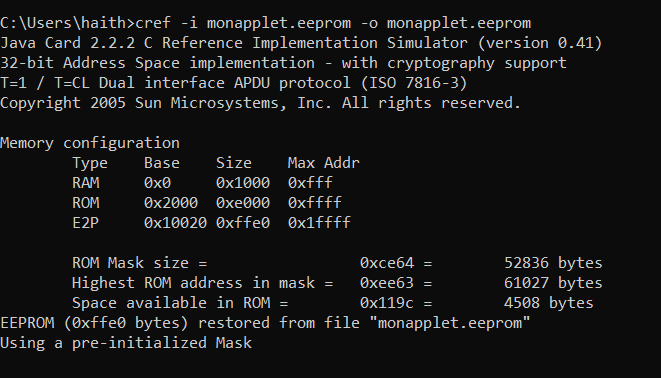


**Remarque :**

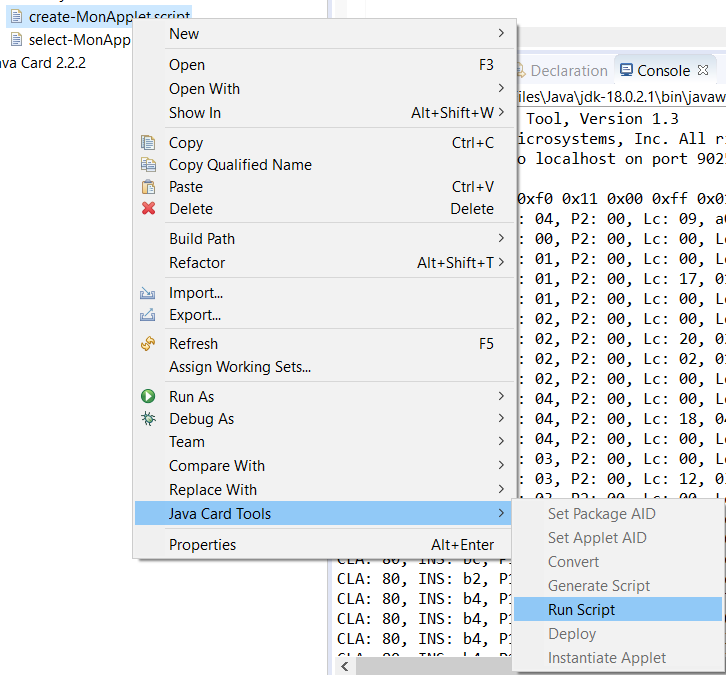


1. Relançons CREF afin d'installer notre applet, en tapant la commande suivante :

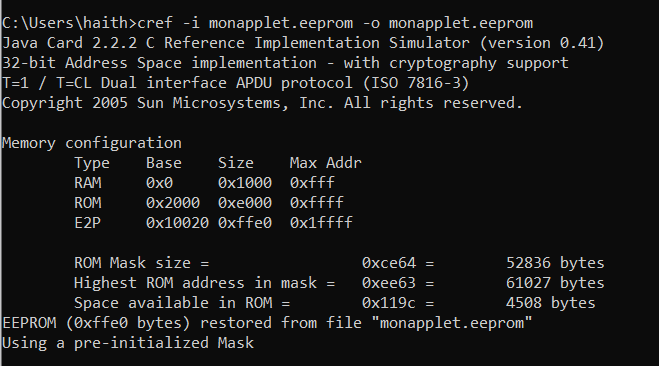
cref -i monapplet.eeprom -o monapplet.eeprom



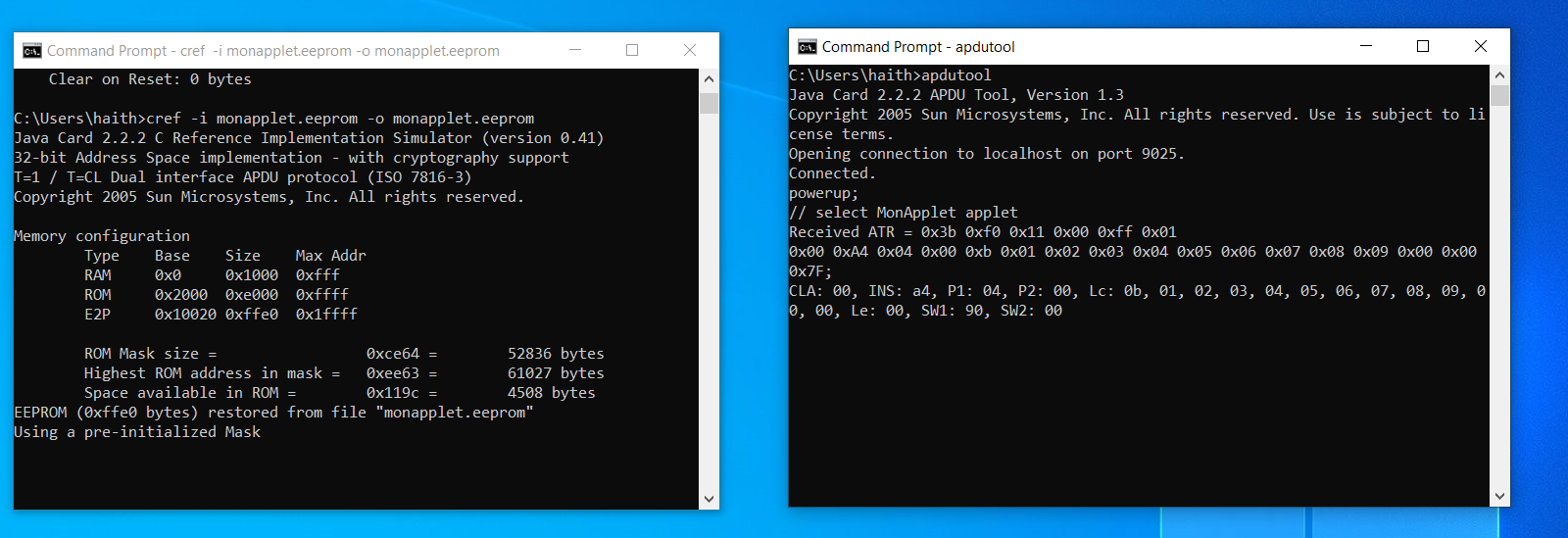
1. Installons maintenant notre applet. Pour cela, dans Eclipse, clic droit sur le script **create-MonApplet.script**, **JavaCard Tools**>**Run Script** :



1. 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**) :



1. Puis, dans un autre terminal, lançons **apdutool**, sélectionnons notre applet, après nous pouvons envoyer des **APDU** à notre applet :

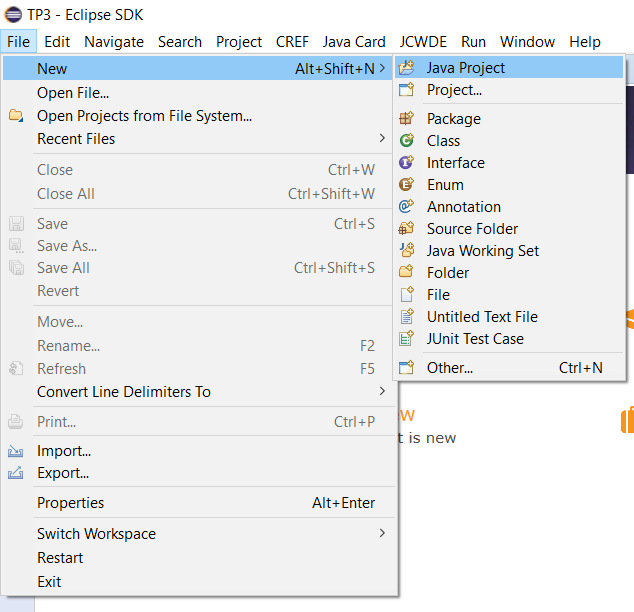


Chapitre III. Programmation d’une application coté client

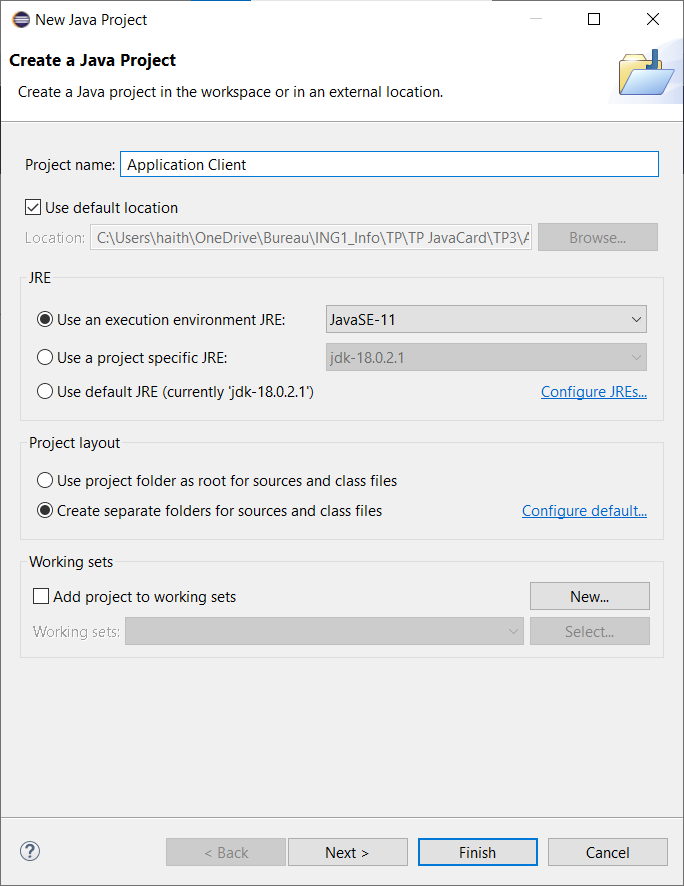
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. Création de l’application client sous Eclipse :
     1. Création d’un nouveau projet Java :

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



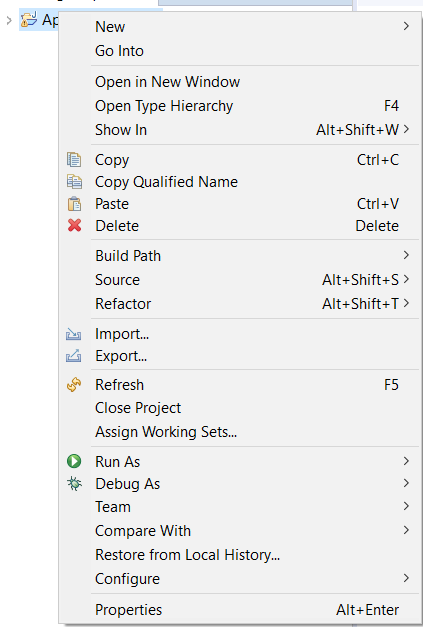
1. Une fenêtre sera affichée, Donner un **nom au nouveau projet** (**Application client** par exemple) puis cliquer sur **Finish** :



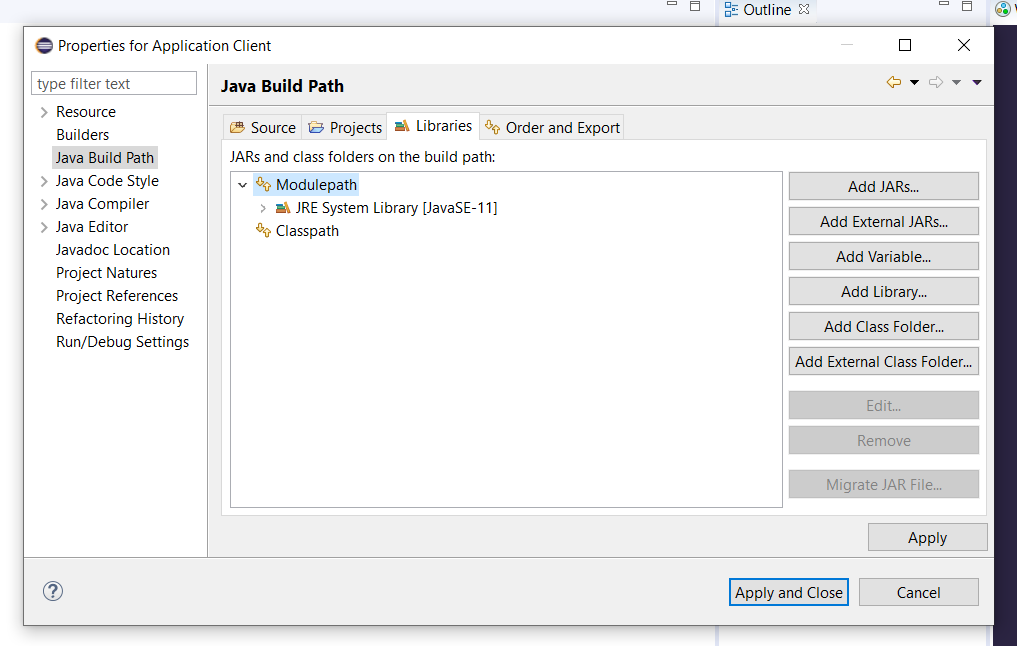
1. Maintenant le Projet est créé avec succès:
   * 1. Ajout de la librairie « apduio » dans le classpath :

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**).

1. Pour cela, faire un **clic droit** sur notre projet **Application Client** > **Propriétés...** :



1. Dans la section **Java Build Path**, sélectionner l'onglet **Librairies > Modulepath** et cliquer sur le bouton **Add External Jars ...** :



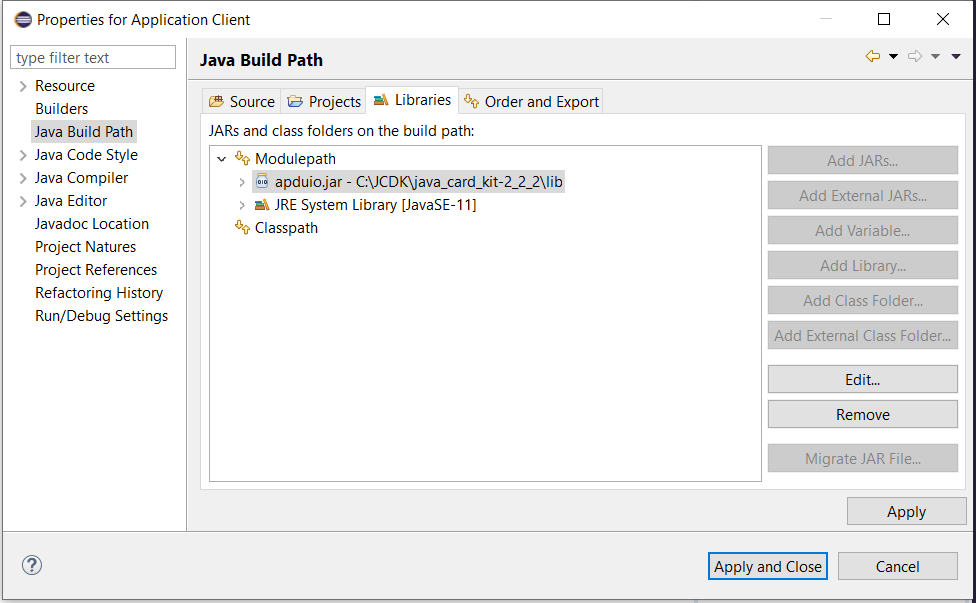
4

3

2

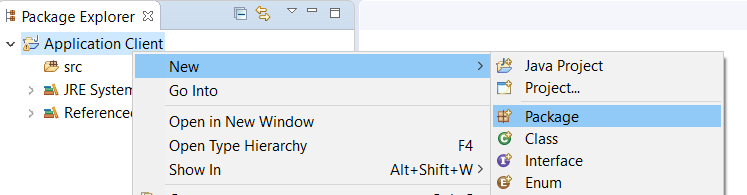
1

1. Sélectionner alors le fichier **apduio.jar** (sous la répertoire **C:\JCDK \java\_card\_kit-2\_2\_2\lib)**, **valider** et appuyer sur le bouton **OK**.

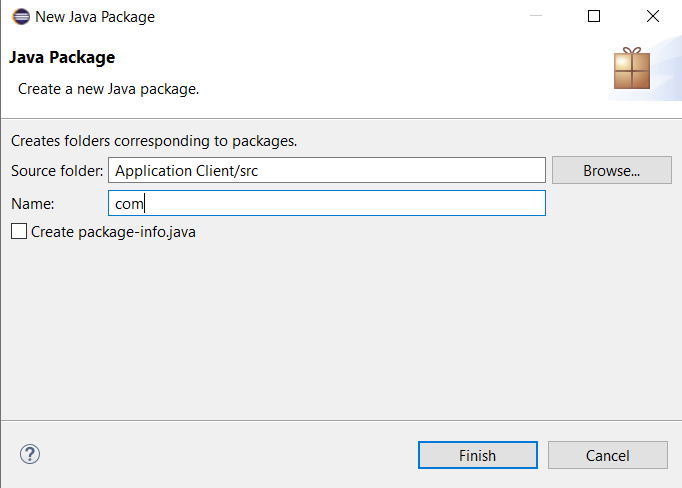


* + 1. Création de la classe principale :

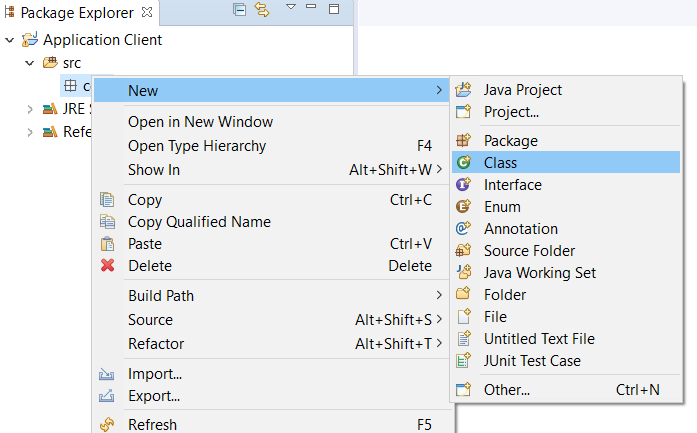
1. Tout d'abord, créons un **nouveau package**. Pour cela, faire un **clic droit** sur notre  
   projet, puis **New** et **Package** :



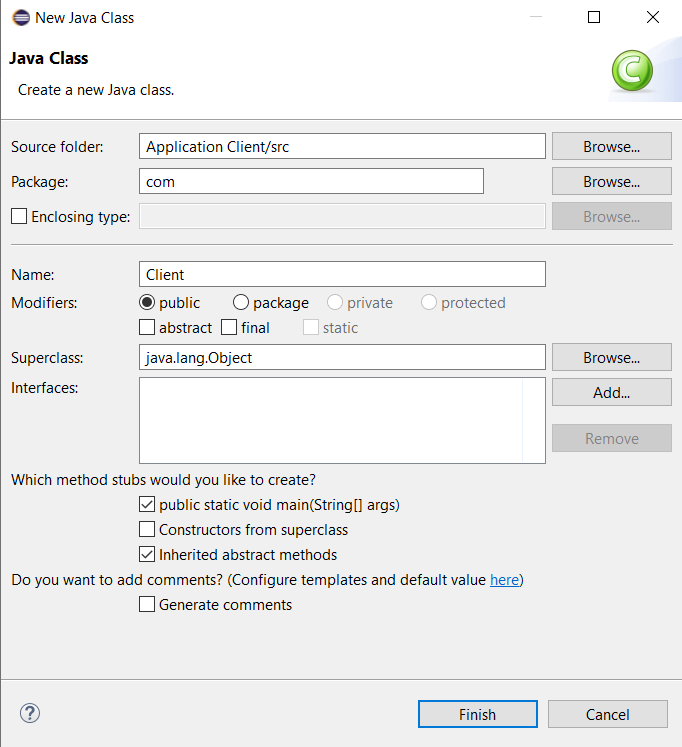
1. Donner **un nom au package** (**com** par exemple ), puis valider à l'aide du  
   bouton **Finish** :

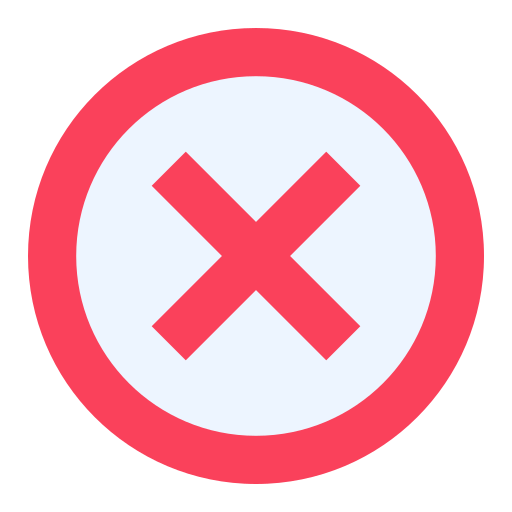


1. Créons maintenant la classe principale de notre application. Pour cela, faire un **clic droit** sur le package créé puis **New** > **Class** :

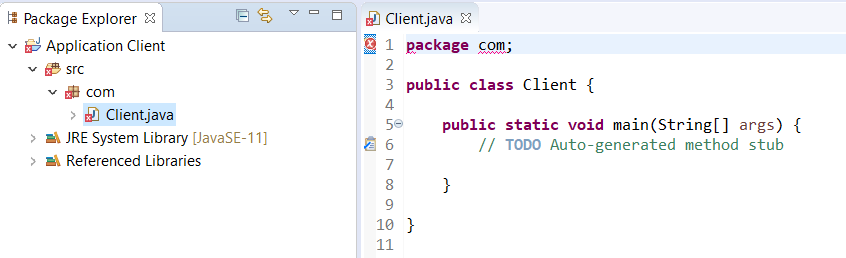


1. Donnons un **nom** à notre nouvelle classe (**Client** par exemple), cocher la case **public** **static void main** puis cliquer sur Finish :



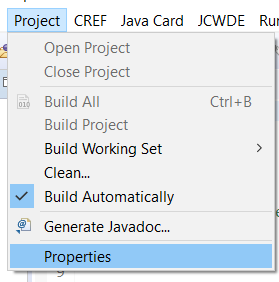


Si une erreur parait comme suit, Voici les étapes qu’il faut suivre !!

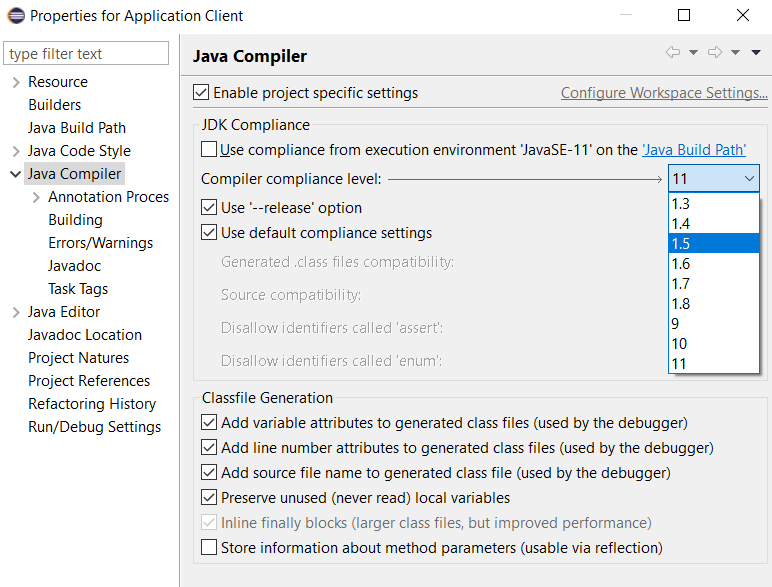


**Erreur !!!**

1. Aller dans le menu barre, sélectionner **Project** > **Properties**



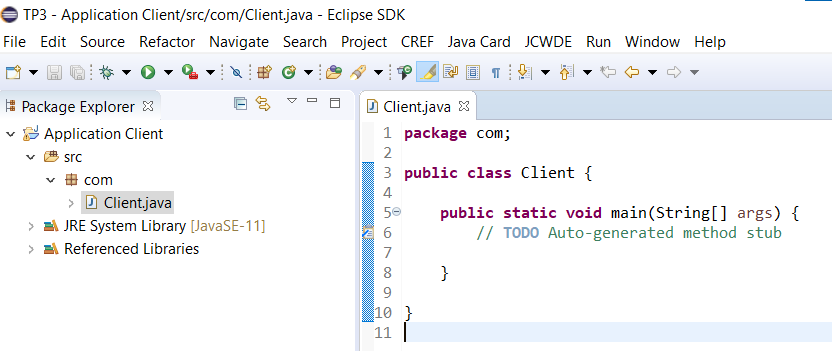
1. Cliquer sur **Java Compiler**, puis **décocher** l’option **« Use Compliance from execution… »,** Enfin changer le **Compiler compliance level** à **1.5** puis **valider**



**Décocher cette option**



Voila le problème est résolu

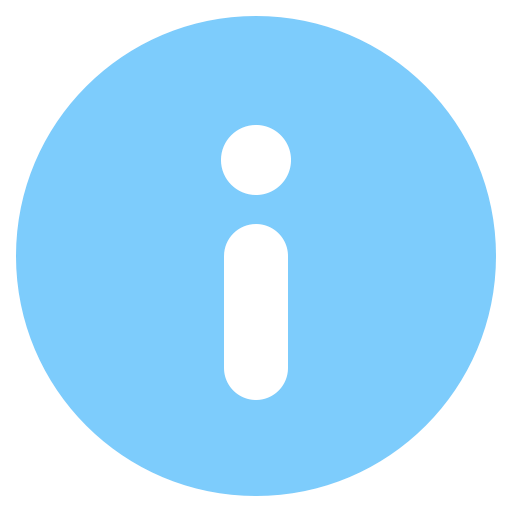


* + 1. Implémentation de l’application client :

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 :

1. **Etape 1 : Connexion :**

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



**C’est quoi un socket Java ?**

Un socket correspond à un port d'écoute sur une machine (cliente / serveur) et servent donc à pouvoir échanger des données entre plusieurs ordinateurs connectés au réseau. Les sockets constituent une API permettant aux applications d'accéder au réseau pour communiquer.

***Les Importations :***

**package** com;

**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** Client {

/\* Déclaration des variables \*/

**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;

**public** **static** **void** main(String[] args) **throws** IOException, CadTransportException {

/\* 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** (Exception e) {

System.***out***.println("Erreur : impossible de se connecter a 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 a la Javacard");

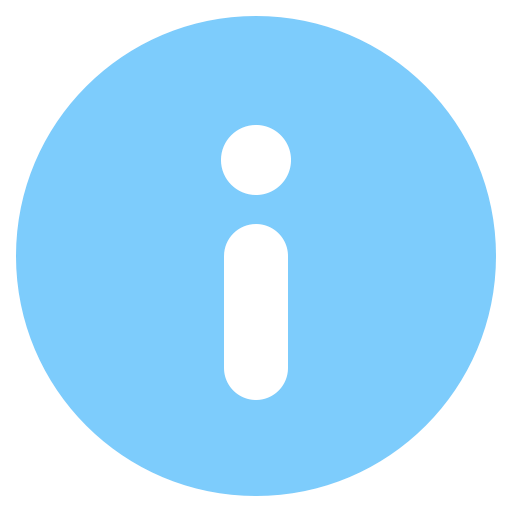
**return**;

}

**Etape 2 : Sélection :**

**Comment sélectionner un applet ?**

Une applet Java Card doit supporter un ensemble de commandes APDU, comprenant la commande SELECT APDU et une ou plusieurs commandes de traitement d'APDUs. **La commande SELECT invite le JCRE à sélectionner une applet sur la carte**.



/\* 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);

cad.exchangeApdu(apdu);

**if** (apdu.getStatus() != 0x9000) {

System.***out***.println("Erreur lors de la sélection de l'applet");

System.*exit*(1);

}

1. **Etape 3 : Invocation des services implémentés**

/\* 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 - Inrementer 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 = System.***in***.read();

**while** (!(choix >= '1' && choix <= '5')) {

choix = System.***in***.read();

}

apdu = **new** Apdu();

apdu.command[Apdu.***CLA***] = Client.***CLA\_MONAPPLET***;

apdu.command[Apdu.***P1***] = 0x00;

apdu.command[Apdu.***P2***] = 0x00;

apdu.setLe(0x7f);

**switch** (choix){

**case** '1':

apdu.command[Apdu.***INS***] = Client.***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***]=Client.***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***] =Client.***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***]= Client.***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':

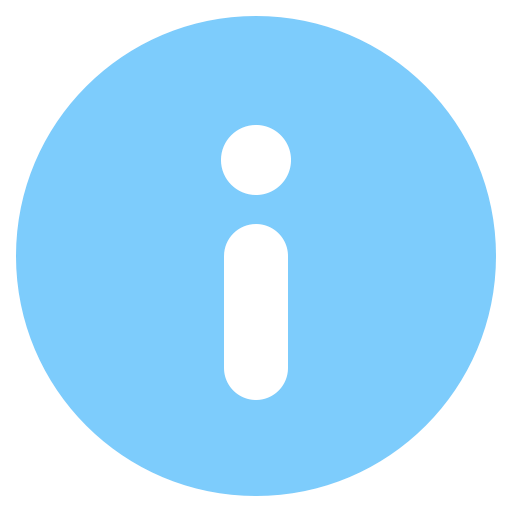
fin = **true**;

**break**;

}

}// fermeture du boucle while

1. **Etape 4 : Mise hors tension :**



**Comment déconnecter de la JavaCard :**

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**;

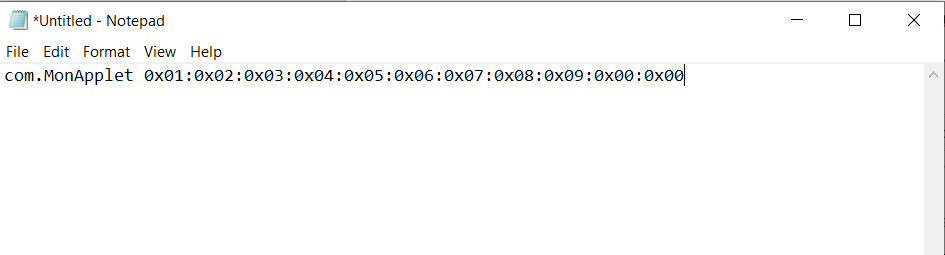
}

* + 1. 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.

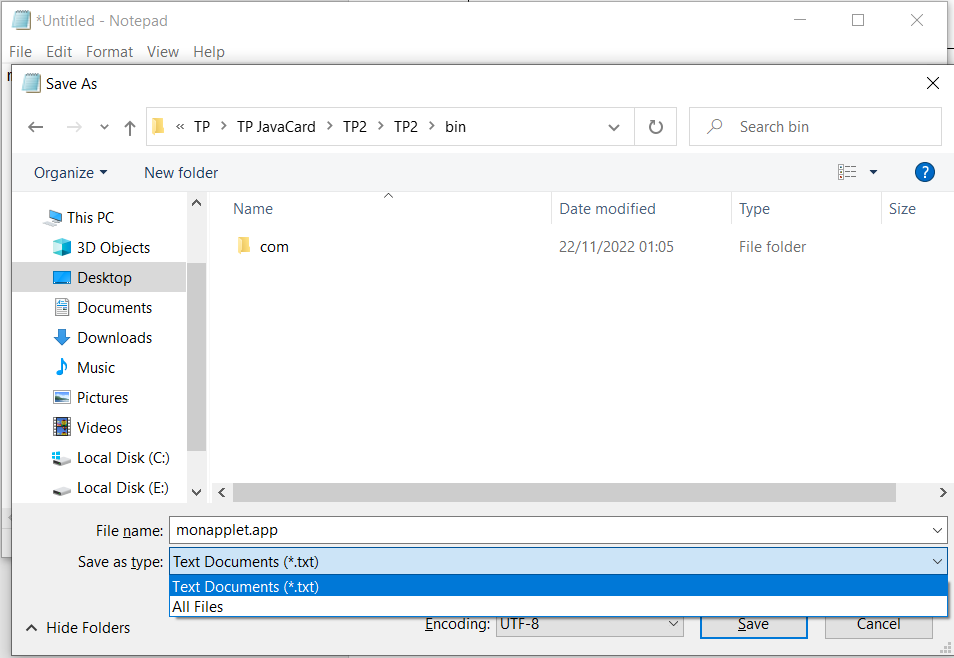
1. Créons notre fichier que nous appellerons "**monapplet.app**" (répertoire parent du  
   package contenant le fichier class de l'applet card **« .\workspace\TP2\bin »**) . Il  
   contiendra la ligne suivante :

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



Ouvrir un éditeur du texte (Par exemple : Notepad) et coller le code ci-dessus. Avec la vérification du **nom du package** et le **nom de la classe**

1



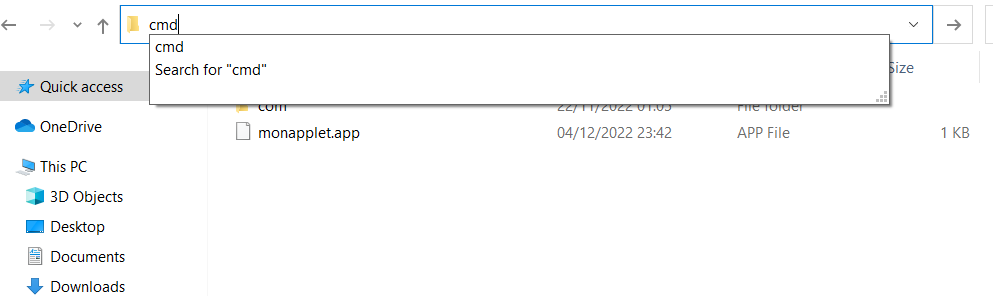
3

Ecrire le **nom du fichier** avec l’extension **.app** (Ici **monapplet.app**) puis changer le type du fichier avec « **All Files**». Enfin cliquer sur **Save.**

Cliquer sur **Ctrl+s** puis déplacer dans le répertoire **«\workspace\TP2\bin »**

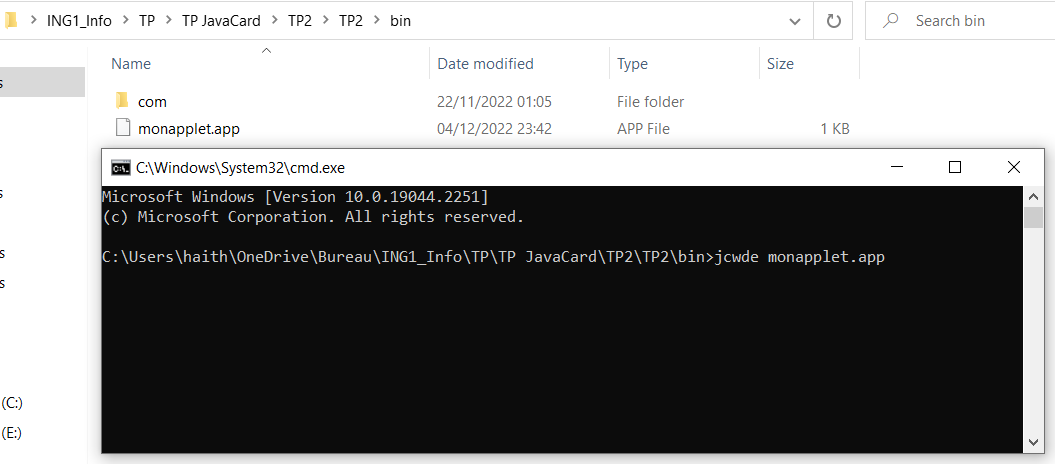
2

1. 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\TP2\bin »**) :



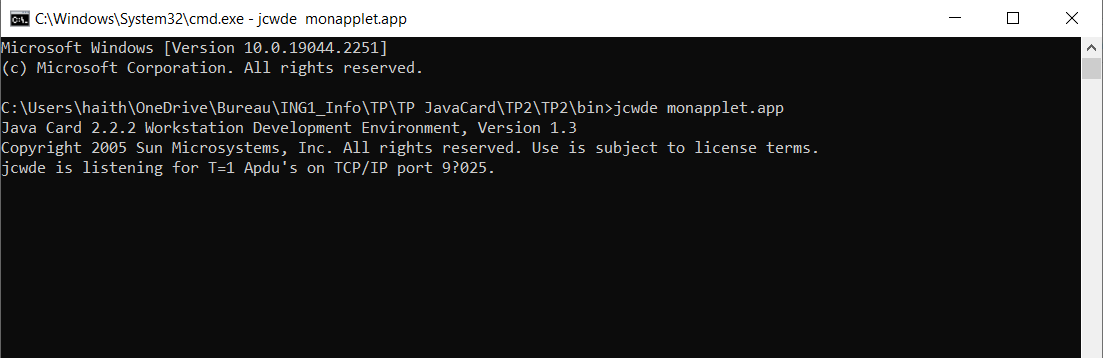
Déplacer dans le répertoire **« .\workspace\TP2\bin »** et dans la barre qui contient le chemin du dossier Ecrire **cmd** puis **Valider**

1



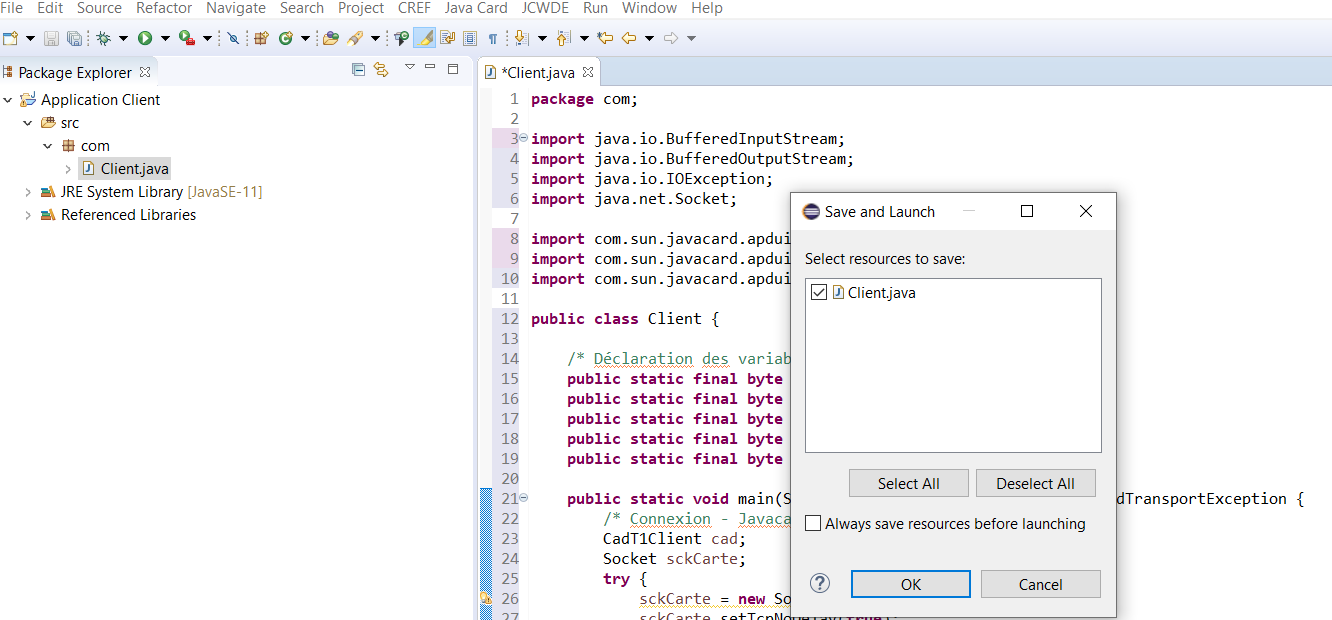
Exécutez alors la commande « **jcwde monapplet.app** »

2

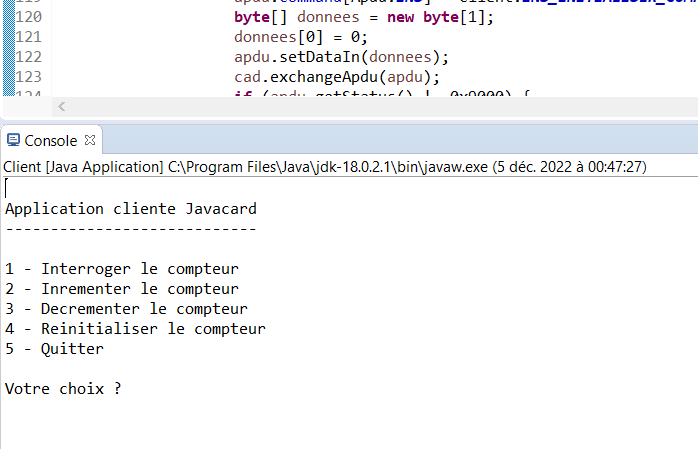


Voici le Message du succès.

1. Maintenant que notre simulateur est lancé, **lançons notre application cliente** :

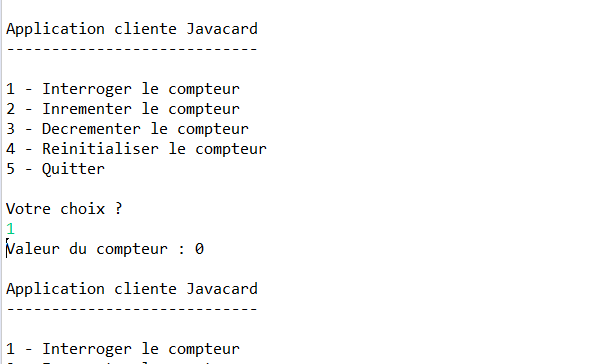


1



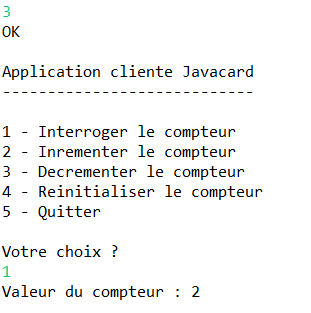
**Succès**

1. Commençons par **interroger** le compteur : tapons **1** puis validons :



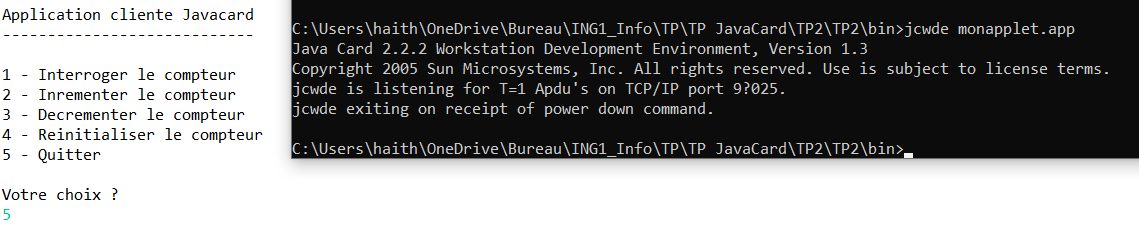
**Le compteur est bien à 0.**

1. **Incrémentons** maintenant le compteur **3 fois** puis **décrémentons-le une fois**. Lorsque nous réinterrogeons la carte :



**Le compteur vaut 2 donc tout va bien.**

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



**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.

Chapitre IV. Réalisation du Mini-Projet

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. Interface Graphique ( Partie Client ) :

Pour réaliser l’interface graphique du Partie Client, on va utiliser Java Swing.

Il existe deux méthodes pour la réalisation du design de cette interface :

* Code pure
* En utilisant la palette (Drag and Drop)
* Dans ce projet on va construire l’interface en utilisant la palette. Pour cela on va installer le plugin **WindowBuilder** dans eclipse.
  + 1. WindowBuilder Introduction :

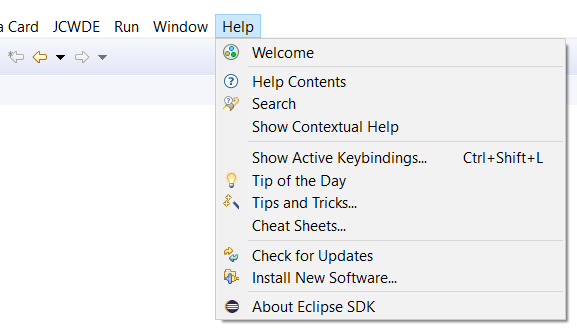
Le plugin Eclipse WindowBuilder est un concepteur Java GUI visuel, puissant et facile à utiliser permettant la création d'applications GUI Java sans vous casser la tête à écrire du code pour afficher des objets graphiques simples comme fenêtres, bouton de commandes, champs de textes... Avec WindowBuilder, vous pouvez créer des fenêtres compliquées en quelques minutes, il suffit d’utiliser le concepteur visuel et le code Java sera automatiquement généré pour vous. Vous pouvez facilement ajouter des contrôles à l'aide de glisser-déposer, gérer les événements de vos contrôles, modifier diverses propriétés des contrôles à l'aide d'un éditeur de propriétés et bien plus encore. Le code généré ne nécessite aucune bibliothèque personnalisée supplémentaire pour compiler et exécuter : l'ensemble du code généré peut être utilisé sans installer WindowBuilder.

L'éditeur est doté des principaux composants interface utilisateur (user interface ) suivants:  
**•    Design View** - la zone de présentation visuelle principale.  
•    **Source View**- code d'écriture et analyse du code généré  
•    **Structure View**- composée de l'arbre de composant et du volet Propriété.  
•    **Component Tree**- montre la relation hiérarchique entre tous les composants.  
•    **Property Pane**- affiche les propriétés et les événements des composants sélectionnés.  
•    **Palette** - permet un accès rapide aux composants spécifiques à une trousse d'outils.  
•     **ToolBar** - permet d'accéder aux commandes couramment utilisées.  
•     **Context** Menu - permet d'accéder aux commandes couramment utilisées.

* + 1. Installation de WindowBuilder :

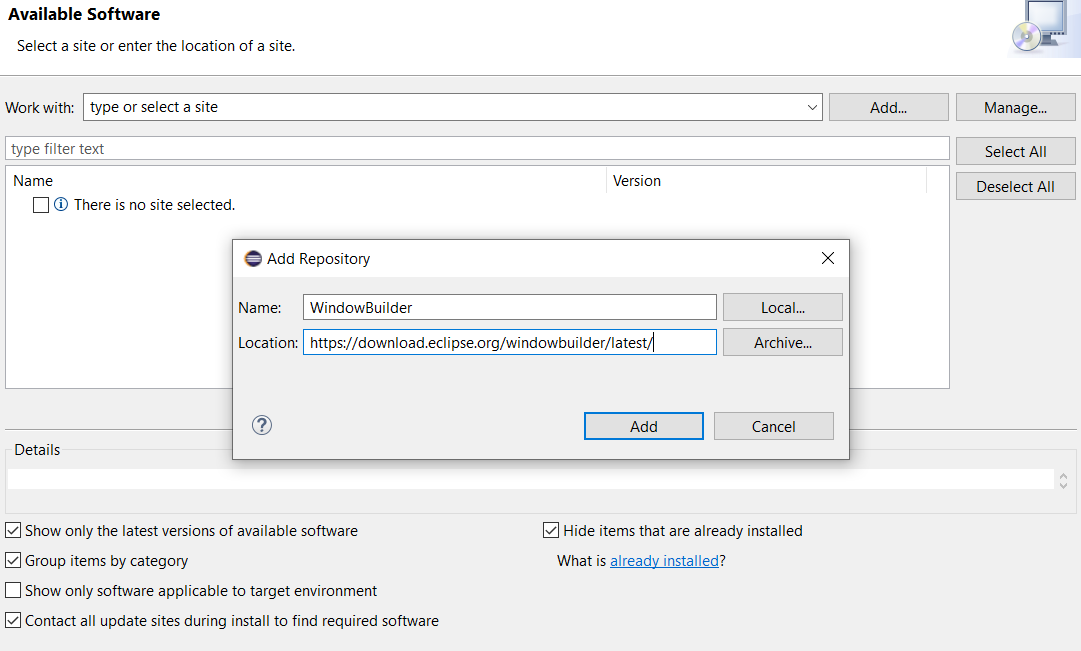
Le plugin Eclipse WindowBuilder est un concepteur Java GUI visuel, puissant et facile à utiliser permettant la création d'applications GUI Java sans vous casser la tête à écrire du code pour afficher des objets graphiques simples comme fenêtres, bouton de commandes, champs de textes... Avec WindowBuilder, vous pouvez créer des fenêtres compliquées en quelques minutes, il suffit d’utiliser le concepteur.

1. Lancer Eclipse puis Depuis le menu Help d’Eclipse, choisissez **Help > Install new Software...**



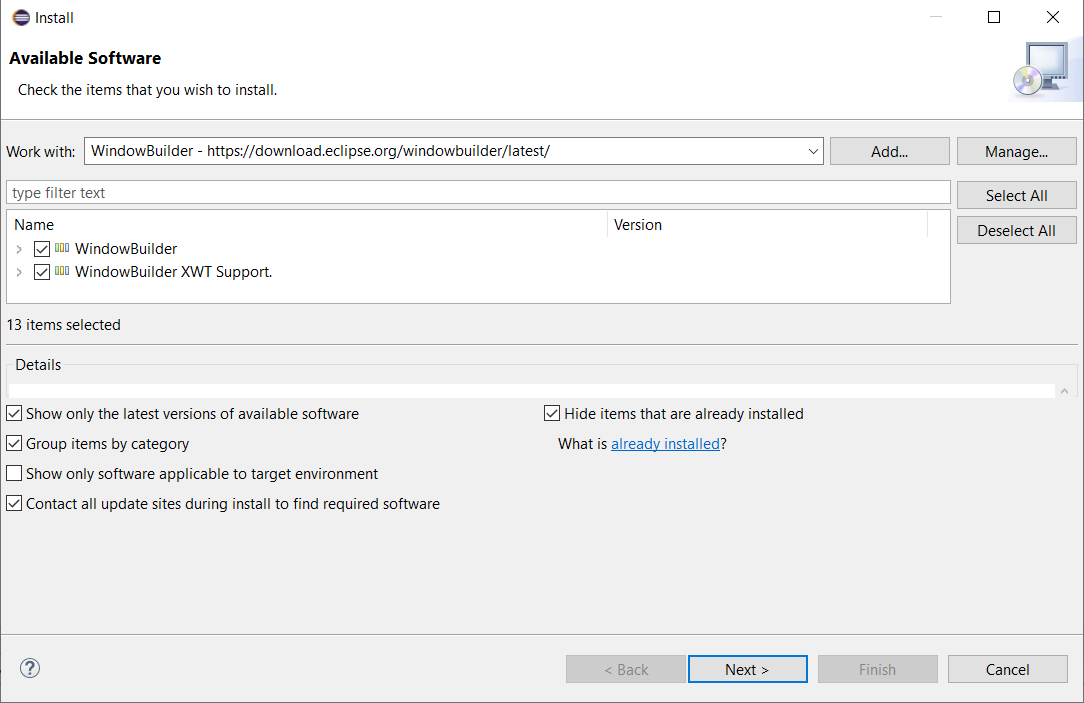
1. Dans la première fenêtre qui apparait cliquez sur **le bouton Add** en suite, dans la boîte de dialogue apparue, dans le champ Name, saisissez un nom descriptif (comme "**WindowBuilder**") et collez le lien ci-dessus dans le champ Emplacement.  Cliquez ensuite sur le bouton **OK** :

**Lien** : <https://download.eclipse.org/windowbuilder/latest/>



1

1. Sélectionnez toutes les **cases à cocher** qui vont apparaître, puis cliquez sur **Suivant** pour installer **WindowBuilder** :

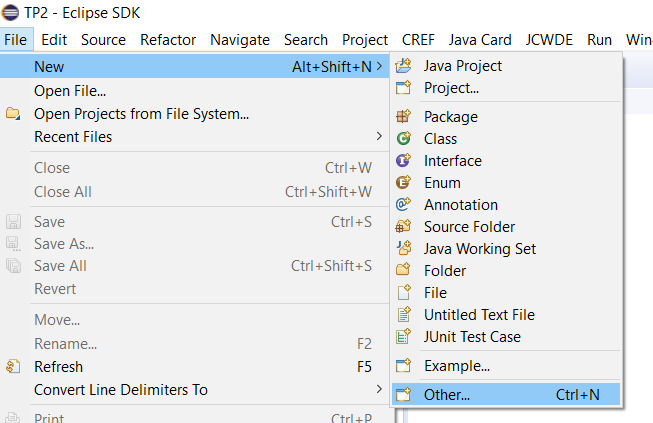


* + 1. Comment utiliser WindowBuilder :

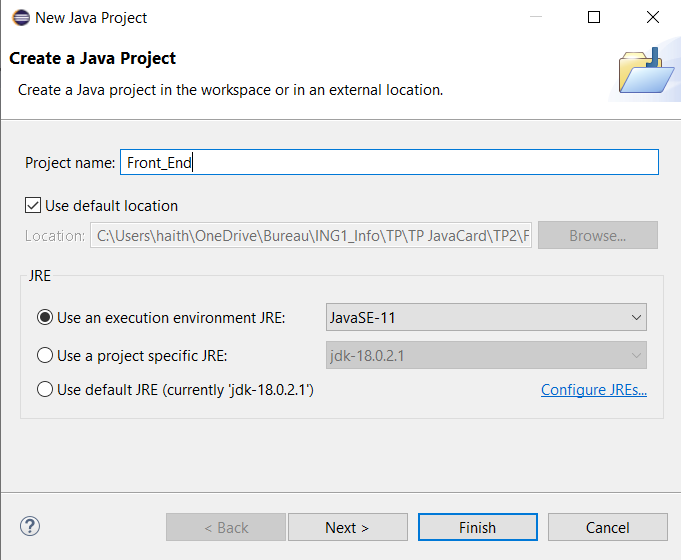
Le plugin Eclipse WindowBuilder est un concepteur

Pour utiliser WindowBuilder :

1. Créer un nouveau projet. Cliquez sur **File > New > Java Project**



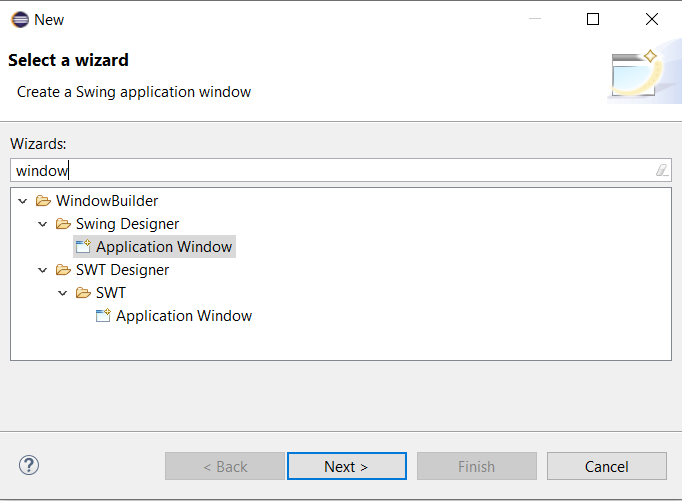
1. Saisir le nom de votre Projet (Ici **Front\_End**) puis cliquer sur **Finish**



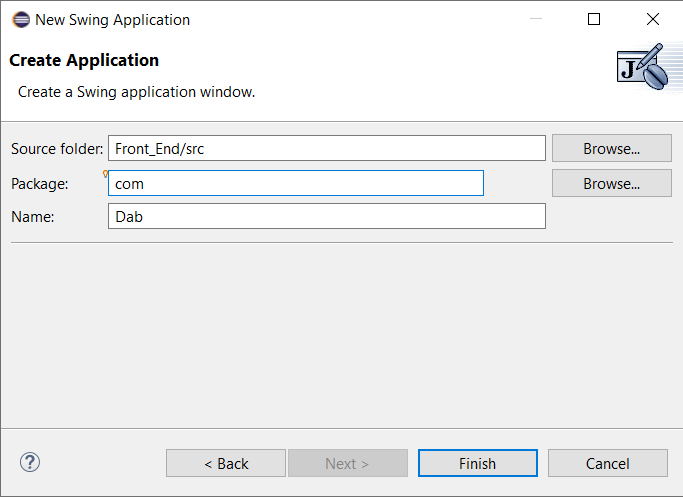
1. Après avoir créer votre Projet Java. Aller sur le dossier clique droit **New**> **Other**



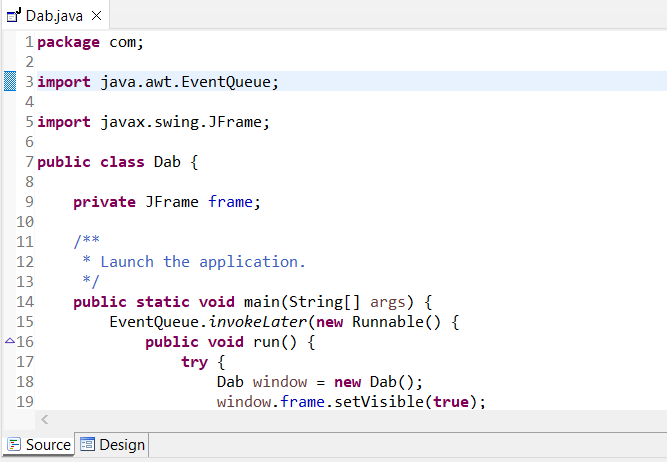
1. Choisir ensuite comme type de projet : **WindowBuilder** -> **Application Window** puis cliquer sur **Next**



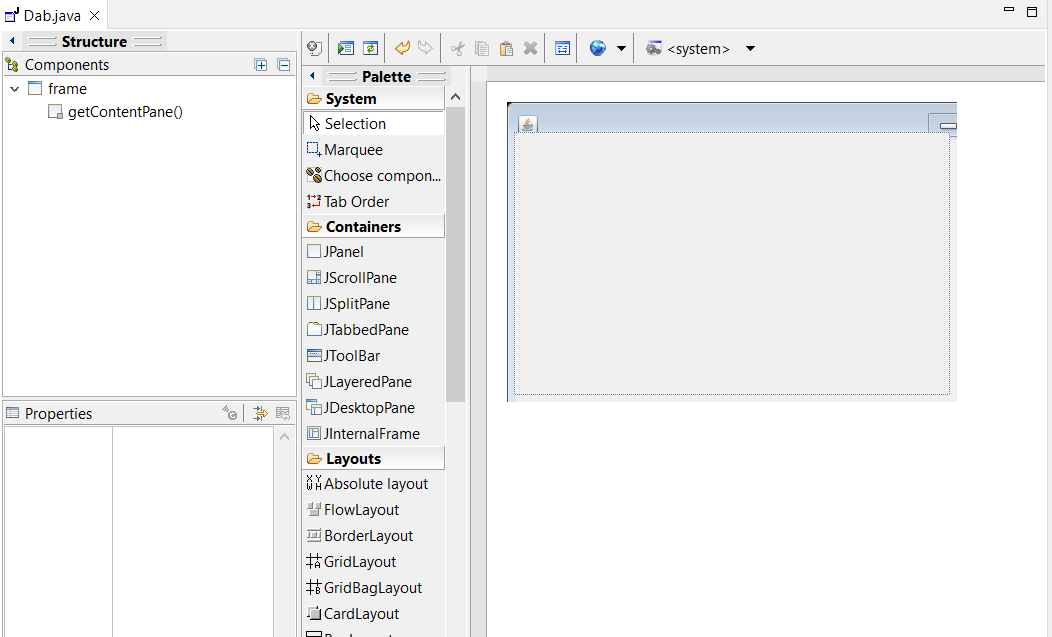
1. Choisir alors le Nom de votre Projet et le package (Ici Name : Front\_End, Package : com).



1. Après avoir cliquez sur le bouton  **Finish** vous obtenez votre **projet WindowBuilder** :



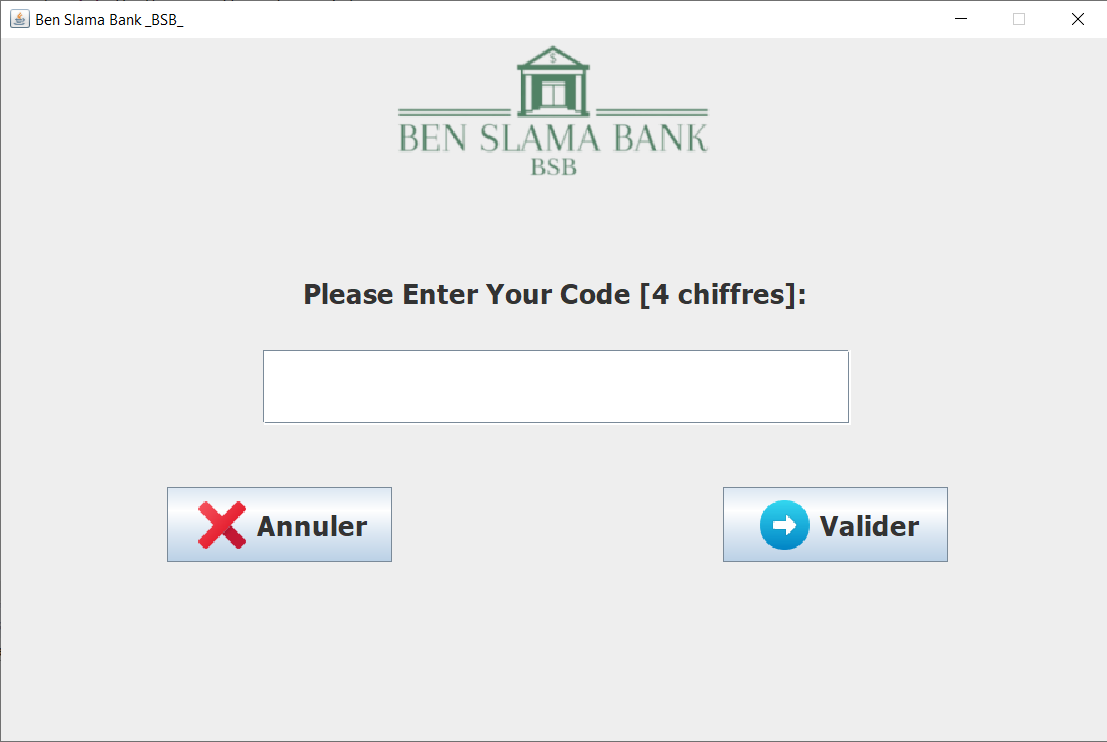
1. Cliquez maintenant sur l'onglet Design pour afficher la fenêtre crée automatiquement par **WindowBuilder** , ainsi que les autres outils graphiques WYSIWYG : Containers, Layouts, Compoasants Swing et Awt ...



**Les outils graphiques de WindowBuilder**

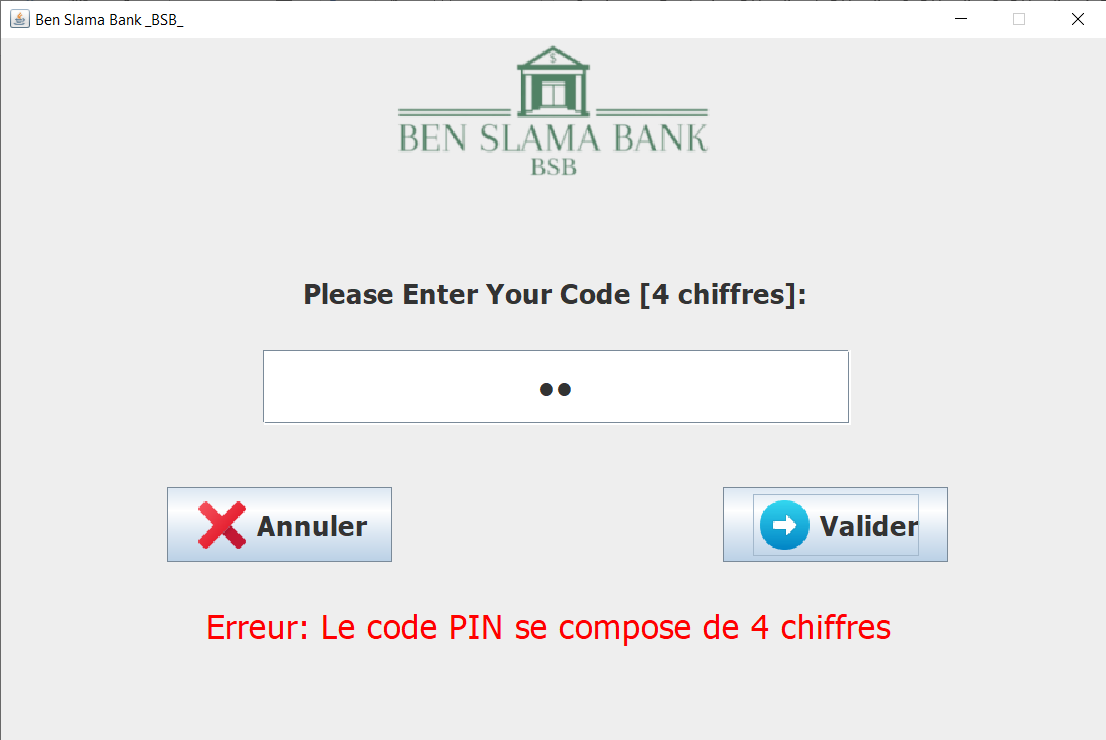
* Maintenant, vous pouvez commencer à construire votre fenêtre et le code sera automatiquement générer dans la partie **code.**
  + 1. Les interfaces de notre application Client :

1. 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.



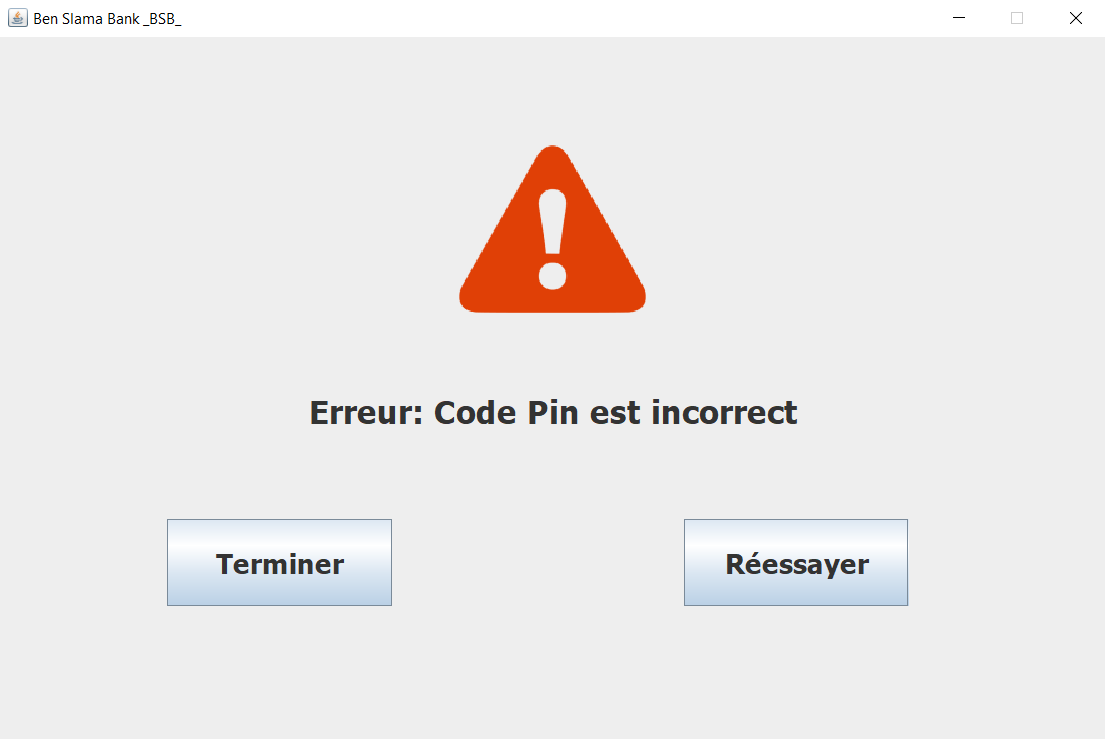
* **1ére Cas :**

Si le texte saisi contient **des lettres ou sa longueur n’est pas égale à 4** un message d’erreur sera affiché dans la même fenêtre.



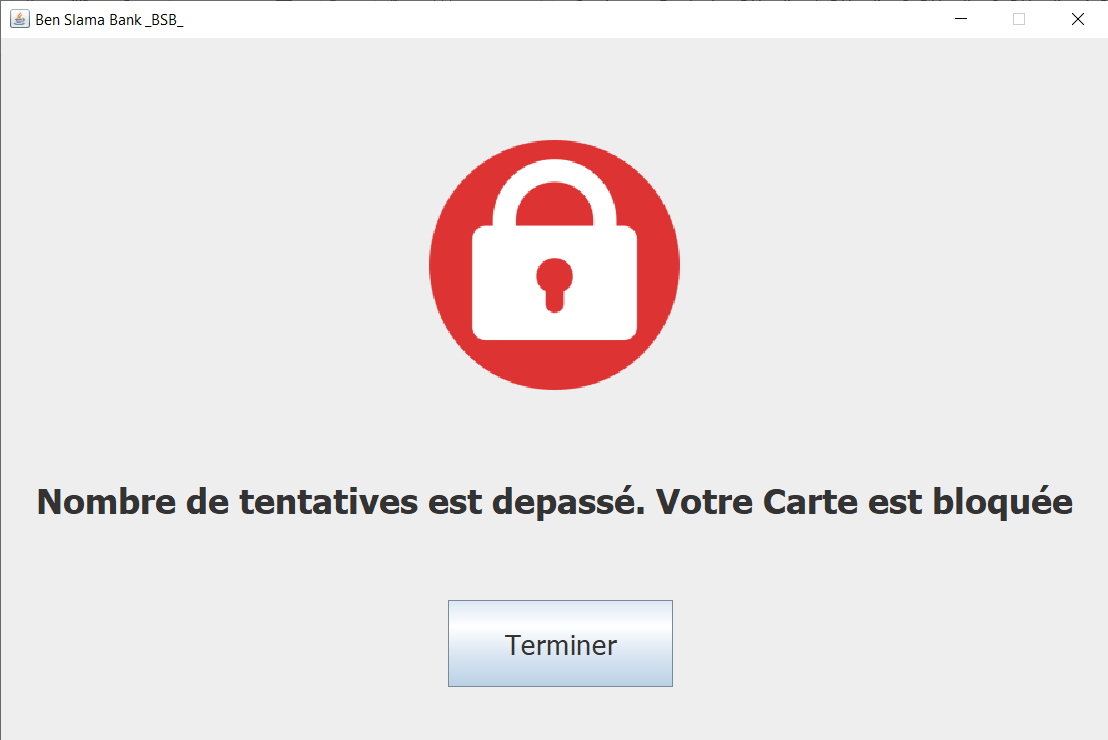
* **2éme Cas :**

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.
* **3éme Cas :**

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



* **4éme Cas :**

Si vous avez saisi le code PIN **correctemen**.



1. La principale fenêtre contient 4 boutons utilisable et 2 autre boutons non utilisable.

* **Solde** : Permet de consulter le solde.
* **Créditer** : Permet de créditer un montant.
* **Débiter** : Permet de débiter un montant.
* **A propos** : Contient mes informations.
* **RFU** : Reserved for Future Use.

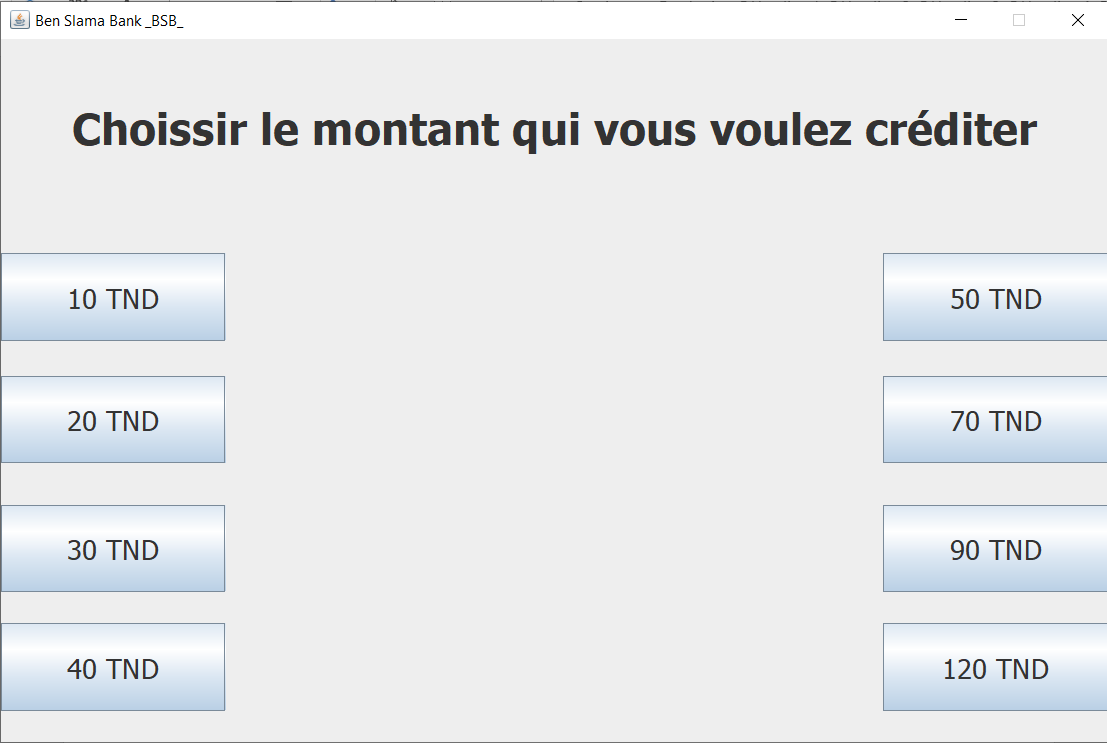


**Fenêtre Principale**

* **Interface Solde:**

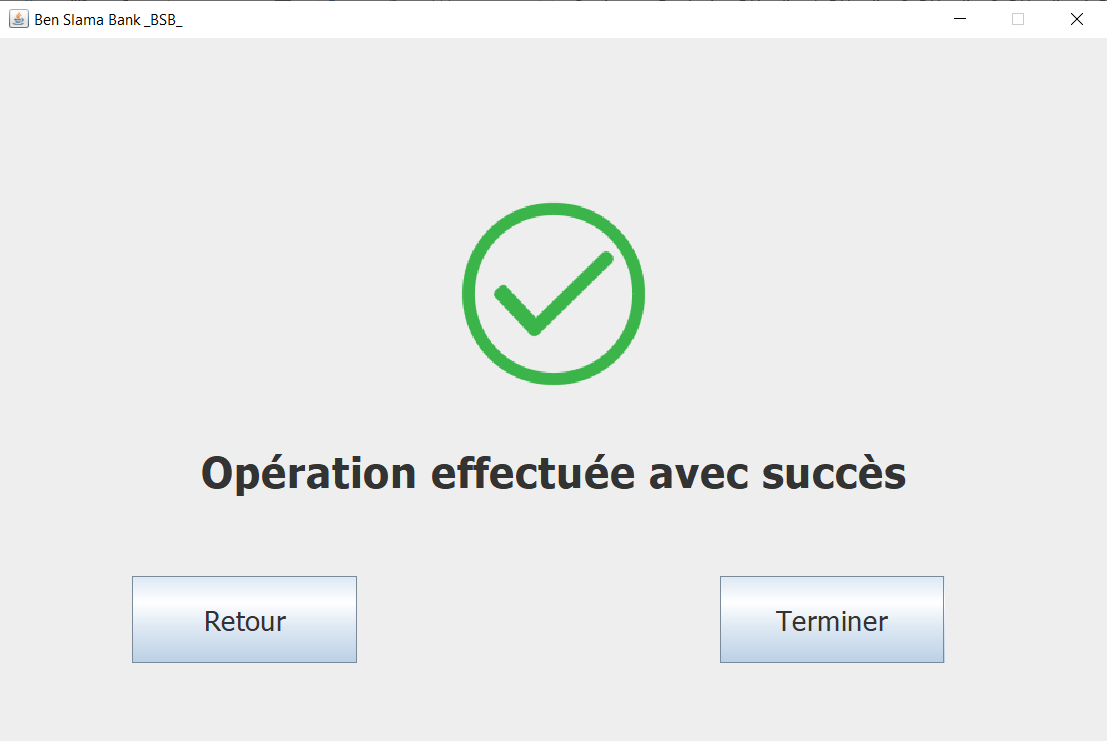


* **Interface Créditer:**

****

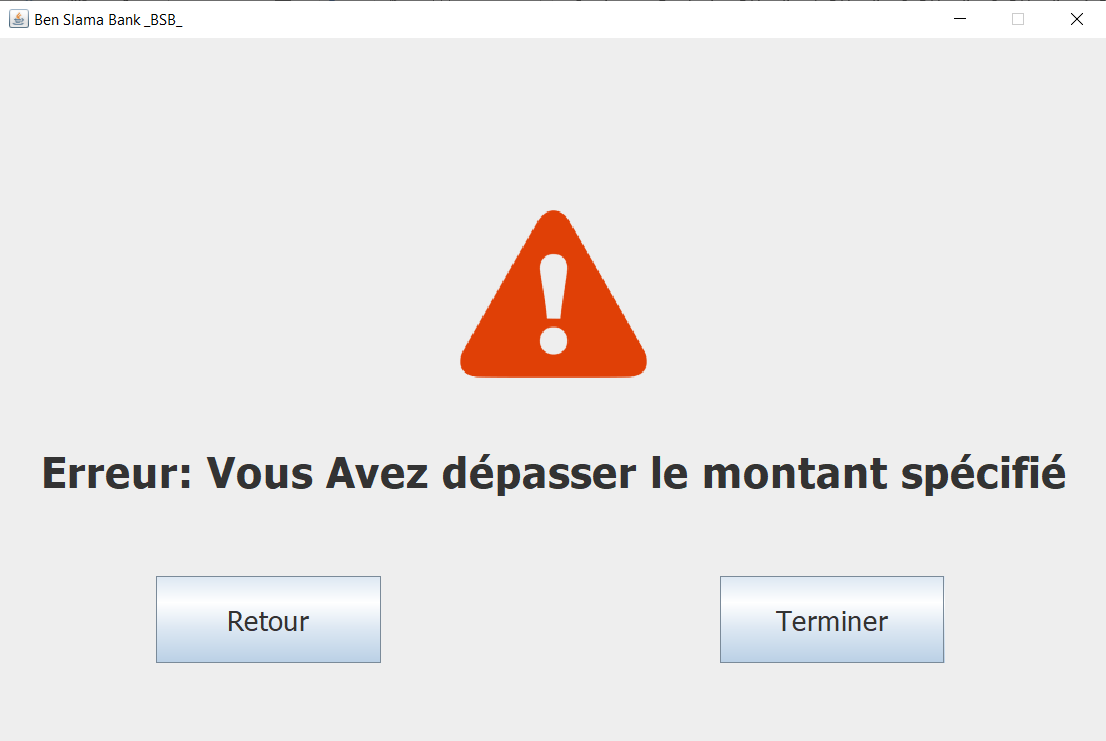
**1ére Cas :**

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

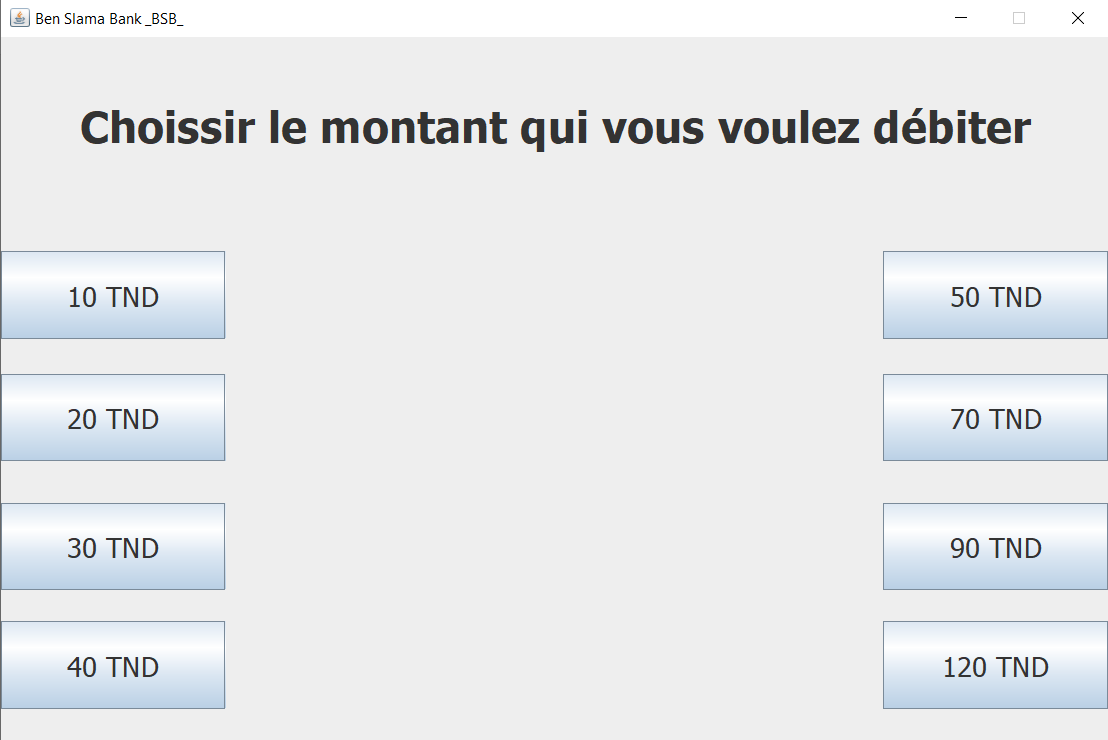


**2ére Cas :**

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

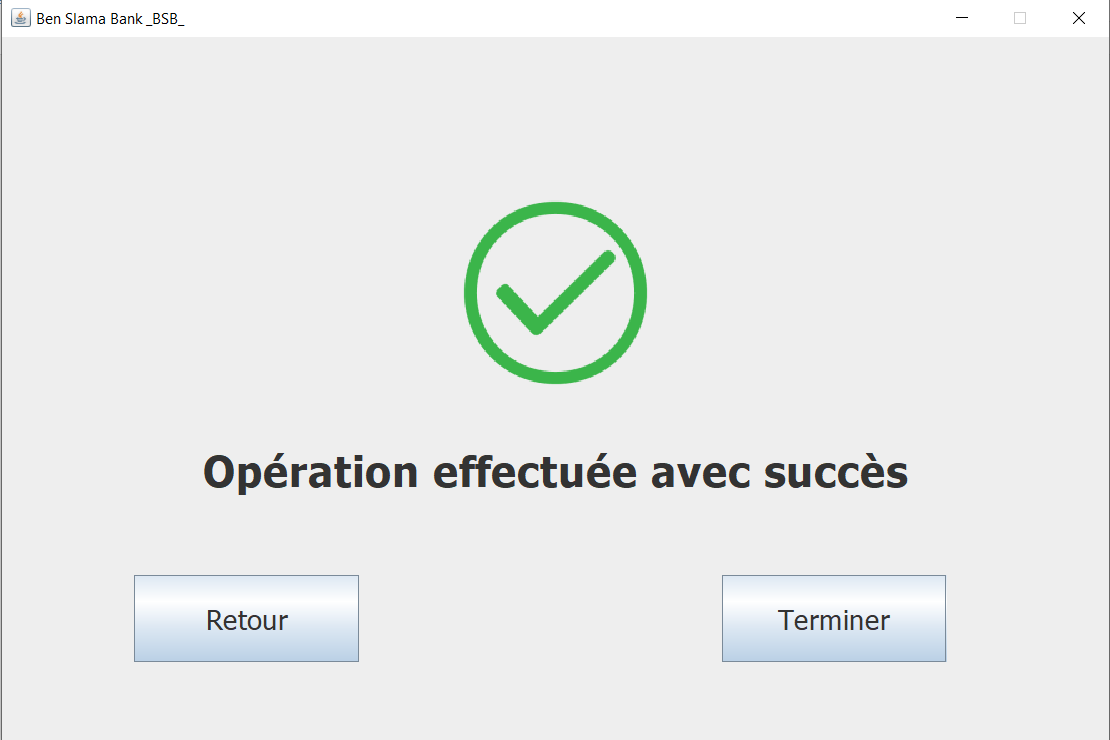


* **Interface Créditer:**

****

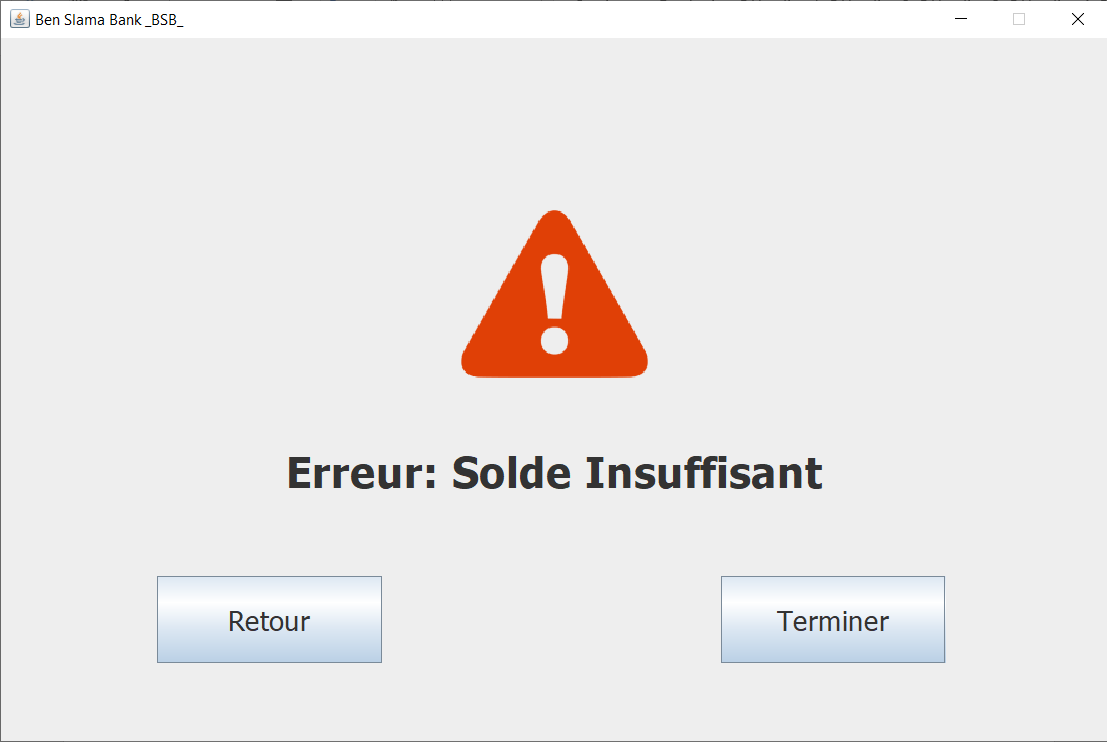
**1ére Cas :**

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

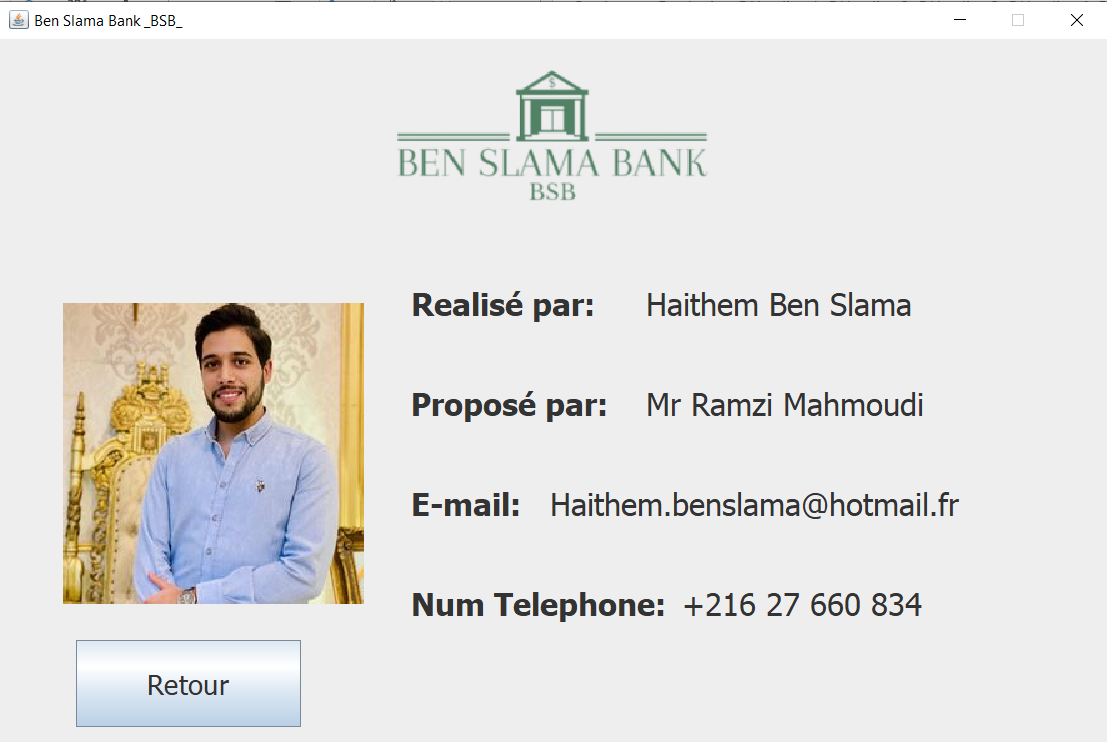
****

**2ére Cas :**

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

****

* **Interface A propos:**

****

* + 1. Implémentation des méthodes nécessaires :

1. **Déclaration des variables :**

/\* variables \*/

**static** Apdu *apdu* ;

**static** CadT1Client *cad*;

1. **Implémentation de la methode Msg() :**

Cette méthode permet l’envoi des Apdus.

**public** Apdu Msg(**byte** ins, **byte** lc, **byte**[] data,**byte** le) **throws** IOException, CadTransportException{

*apdu* = **new** Apdu();

*apdu*.command[Apdu.***CLA***] = (**byte**) 0xB0;

*apdu*.command[Apdu.***P1***] = 0x00;

*apdu*.command[Apdu.***P2***] = 0x00;

*apdu*.command[Apdu.***INS***] = ins;

//apdu.setLe(0x7f);

*apdu*.setLe(le);

**if** (data!=**null**)

*apdu*.setDataIn(data);

*cad*.exchangeApdu(*apdu*);

**return** *apdu*;

}

1. **Implémentation de la methode Connect() :**

Cette méthode permet la connexion avec la carte.

**public** **void** Connect(){

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** (Exception e) {

System.***out***.println("Erreur : impossible de se connecter a 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 a la Javacard");

**return**;

}

}

1. **Implémentation de la methode Select() :**

Cette méthode permet la sélection de notre applet dans la carte.

**public** **void** select() **throws** IOException, CadTransportException{

/\* Sélection de l'applet :création du commande SELECT APDU \*/

*apdu* = **new** Apdu();

*apdu*.command[Apdu.***CLA***] = (**byte**) 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);

*cad*.exchangeApdu(*apdu*);

**if** (*apdu*.getStatus() != 0x9000) {

System.***out***.println("Erreur lors de la sélection de l'applet");

System.*exit*(1);

}

}

1. **Implémentation de la methode Deselect() :**

**public** **void** Deselect(){

/\* 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**;

}

}

* 1. Partie Serveur (Back-End) :
     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** **static** **final** **byte** ***INS\_DECREMENTER\_COMPTE*** = 0x03;

**public** **static** **final** **byte** ***INS\_INITIALISER\_COMPTE*** = 0x04;

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

// le maximum de la balance = 500 TND

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

// maximum montant qu'on peut transiter

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

// maximum de code pin erroner

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

// longeur maximale du 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() :**

**public** **boolean** select() {

// pas de selection si le pin est blocker

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

**return** **false**;

**return** **true**;

}

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

**public** **void** 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***);

}

}

* **Implémentation** **de la méthode credit :**

**private** **void** credit(APDU apdu) {

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

// Lc byte denotes the number of bytes in the

// data field of the command APDU

**byte** numBytes = buffer[ISO7816.***OFFSET\_LC***];

// indicate that this APDU has incoming data

// and receive data starting from the offset

// ISO7816.OFFSET\_CDATA following the 5 header

// bytes.

**byte** byteRead = (**byte**) (apdu.setIncomingAndReceive());

// it is an error if the number of data bytes

// read does not match the number in Lc byte

**if** ((numBytes != 1) || (byteRead != 1))

ISOException.*throwIt*(ISO7816.***SW\_WRONG\_LENGTH***);

// get the credit amount

**byte** creditAmount = buffer[ISO7816.***OFFSET\_CDATA***];

// check the new balance

**if** ((**short**) (*balance* + creditAmount) > ***MAX\_BALANCE***)

ISOException.*throwIt*(***SW\_EXCEED\_MAXIMUM\_BALANCE***);

// credit the amount

*balance* = (**short**) (*balance* + creditAmount);

}

* **Implémentation** **de la méthode getBalance :**

**private** **void** getBalance(APDU apdu) {

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

**short** le = apdu.setOutgoing();

**if** (le < 2)

ISOException.*throwIt*(ISO7816.***SW\_WRONG\_LENGTH***);

apdu.setOutgoingLength((**byte**) 2);

buffer[0] = (**byte**) (*balance* >> 8);

buffer[1] = (**byte**) (*balance* & 0xFF);

Util.*setShort*(buffer, (**short**)0, *balance*);

apdu.sendBytes((**short**) 0, (**short**) 2);

}

* **Implémentation** **de la méthode verify :**

**private** **void** verify(APDU apdu) {

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

// retrieve the PIN data for validation.

**byte** byteRead = (**byte**) (apdu.setIncomingAndReceive());

// check pin

// the PIN data is read into the APDU buffer

// at the offset ISO7816.OFFSET\_CDATA

// the PIN data length = byteRead

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

ISOException.*throwIt*(***SW\_EXCEED\_TRY\_LIMIT***);

**if** (pin.check(buffer, ISO7816.***OFFSET\_CDATA***, byteRead) == **false**)

ISOException.*throwIt*(***SW\_VERIFICATION\_FAILED***);

}

* **Implémentation** **de la méthode debit :**

**private** **void** debit(APDU apdu) {

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

**byte** numBytes = (**byte**) (buffer[ISO7816.***OFFSET\_LC***]);

**byte** byteRead = (**byte**) (apdu.setIncomingAndReceive());

**if** ((numBytes != 1) || (byteRead != 1))

ISOException.*throwIt*(ISO7816.***SW\_WRONG\_LENGTH***);

// get debit amount

**byte** debitAmount = buffer[ISO7816.***OFFSET\_CDATA***];

// check the new balance

**if** ((**short**) (*balance* - debitAmount) < (**short**) 0)

ISOException.*throwIt*(***SW\_NEGATIVE\_BALANCE***);

*balance* = (**short**) (*balance* - debitAmount);

}