Java Card 3 Programming
Michel Koenig

TD 1

Installing and configuring the environment

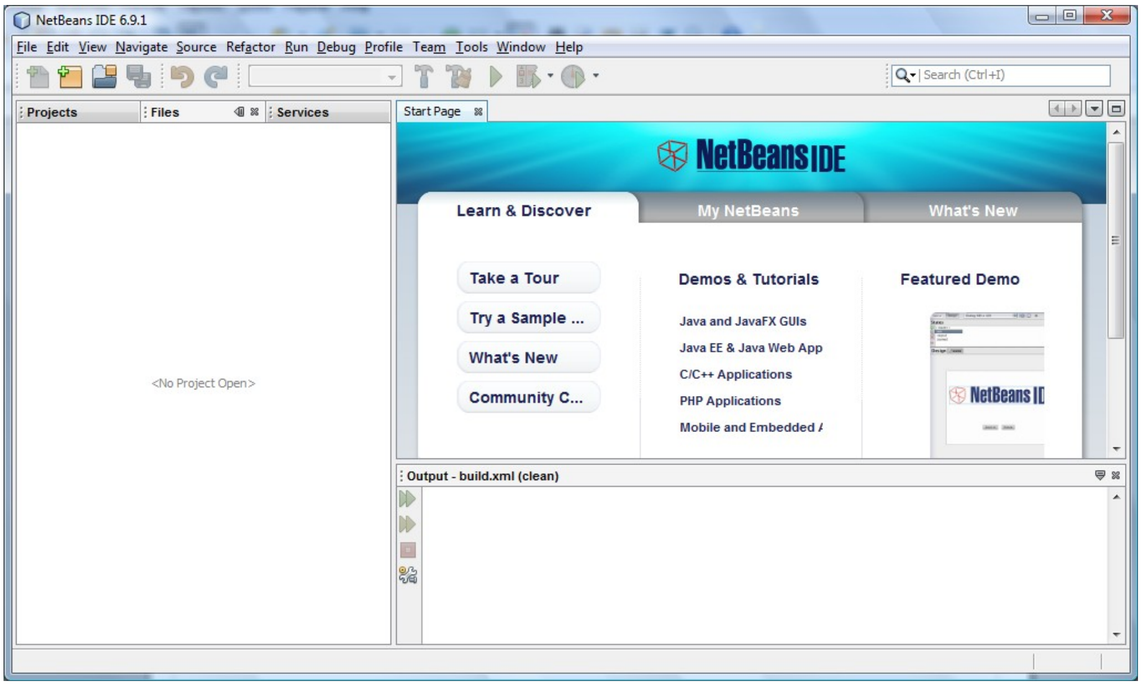
Starting with the 6.9 release, Netbeans involves all the plug-ins needed for Java Card 3 development: wizards are available to create projects for the two flavors of Java Card 3 (connected and classic), simulators are launched automatically when the project is run, etc.

Using your favorite research engine, find the web site for downloading the last release of Netbeans. Several versions are available, select the minimal version which involves the Java Card 3 project development, it will be faster to download. Download and install it with the most standard features.

Netbeans is a very powerful tool but very simple to use. The next paragraph describes the full development of a Java Card 3 project.

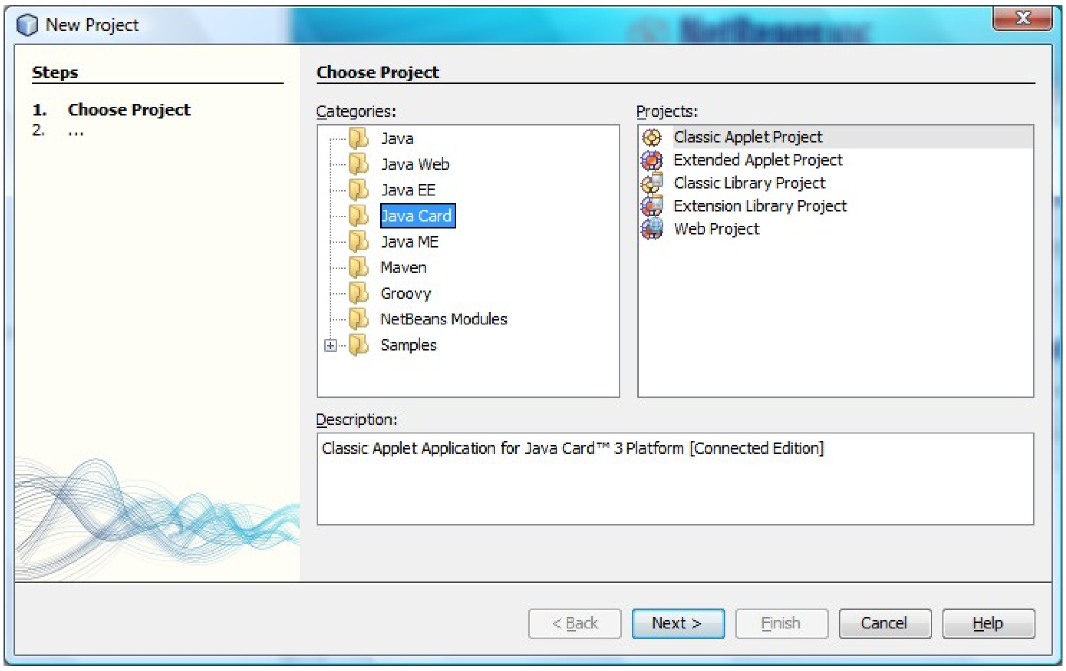
Getting started

After a successful installation, launch Netbeans:



You will discover by yourself all the features of the Netbeans tool. This paragraph focus on the development of an example of an applet, how to make it running and how to debug it.

1. First, create a new project: **select File | New project...**


New Project

Steps

1. Choose Project
2. ...

Choose Project

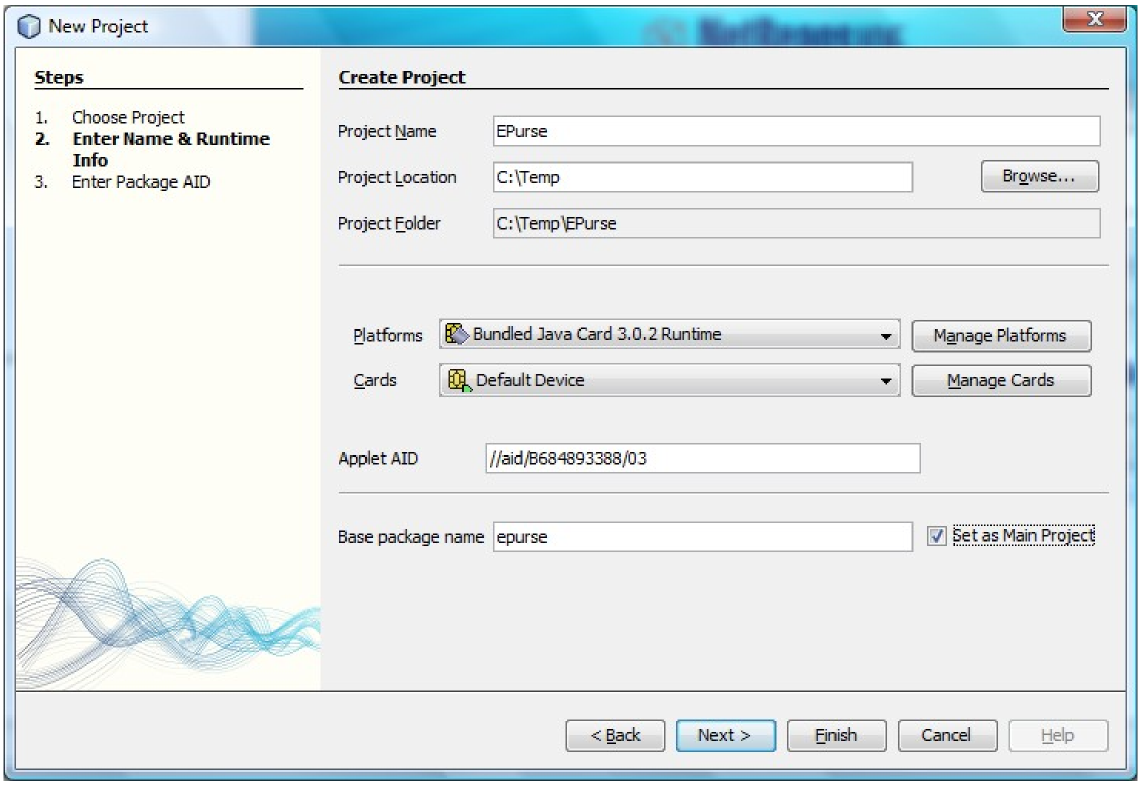
Categories:
- Java
- Java Web
- Java EE
- Java Card
- Java ME
- Maven
- Groovy
- NetBeans Modules
- Samples

Projects:
- Classic Applet Project
- Extended Applet Project
- Classic Library Project
- Extension Library Project
- Web Project

Description:
Classic Applet Application for Java Card™ 3 Platform [Connected Edition]

Netbeans proposes several wizards in order to create projects for various targets. One target is the Java Card. For the Java Card target, Netbeans proposes various types of projects. This first example helps you design and program an e-purse application.

Let's choose a **Classic Applet Project** and click on **Next**:


New Project

Steps

1. Choose Project
2. Enter Name & Runtime Info
3. Enter Package AID

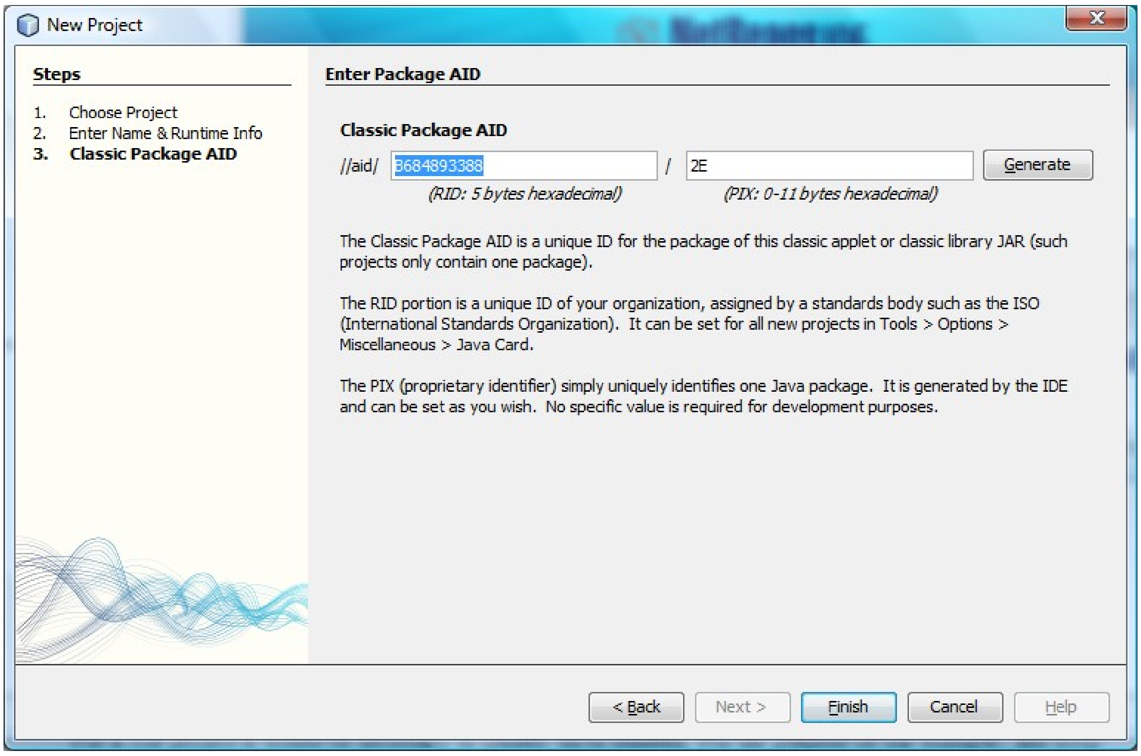
Create Project

Project Name	EPurse
Project Location	C:\Temp
Project Folder	C:\Temp\EPurse
Platforms	Bundled Java Card 3.0.2 Runtime
Cards	Default Device
Applet AID	//aid/B684893388/03
Base package name	epurse

[✓] Set as Main Project

Set the **Project Name** to ***EPurse***, indicate the **Project Location**, set the **Base package name** to ***epurse***, check the box to **Set** the project **as** a **Main Project**, keep the proposed other options. Notice the proposed **Applet aid** (this could be different on your PC).

Click on **Next**:



Classic Package AID

//aid/ 3684893388 / 2E Generate

(RID: 5 bytes hexadecimal) (PIX: 0-11 bytes hexadecimal)

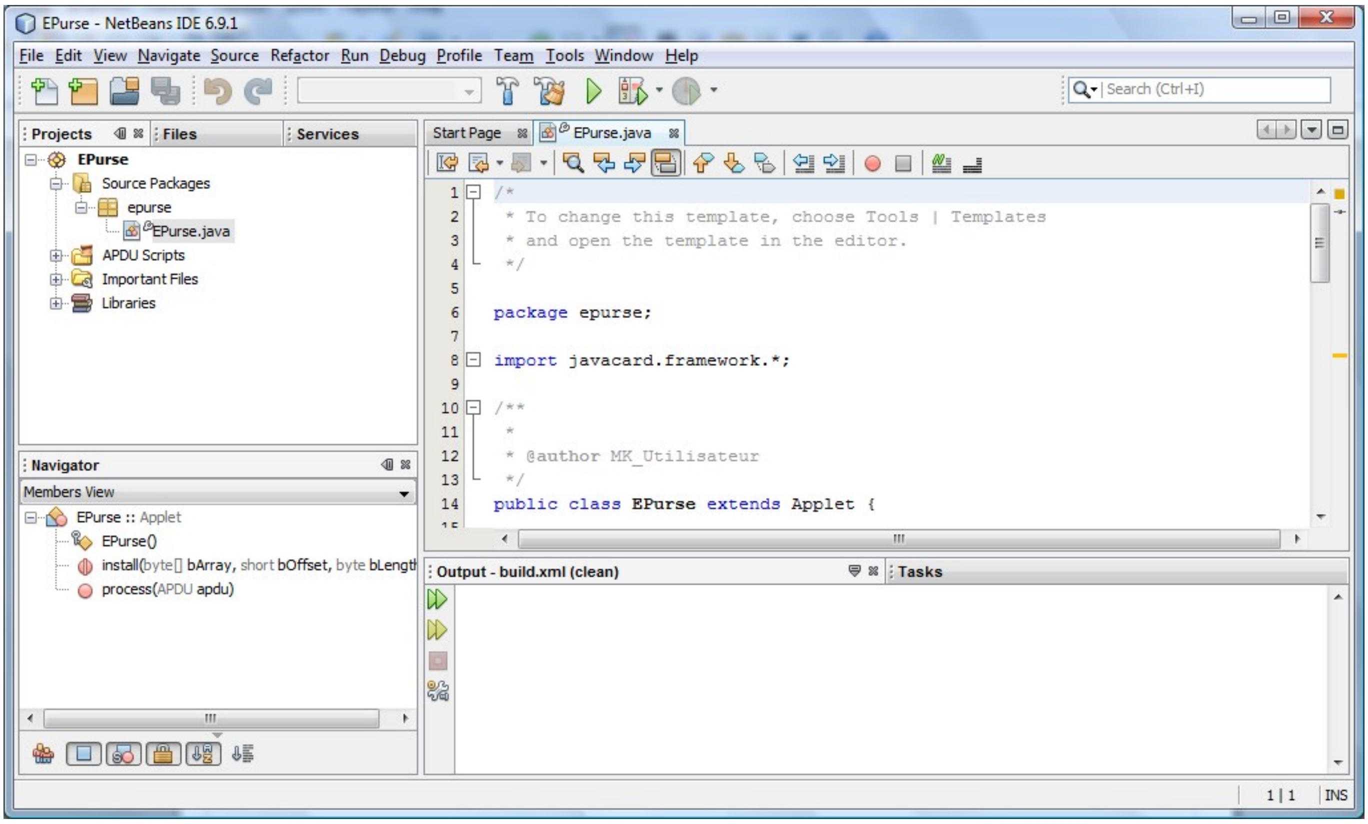
The Classic Package AID is a unique ID for the package of this classic applet or classic library JAR (such projects only contain one package).

The RID portion is a unique ID of your organization, assigned by a standards body such as the ISO (International Standards Organization). It can be set for all new projects in Tools > Options > Miscellaneous > Java Card.

The PIX (proprietary identifier) simply uniquely identifies one Java package. It is generated by the IDE and can be set as you wish. No specific value is required for development purposes.

For a real project it would be necessary to modify these numbers. For the purpose of our example, just keep in mind these numbers.

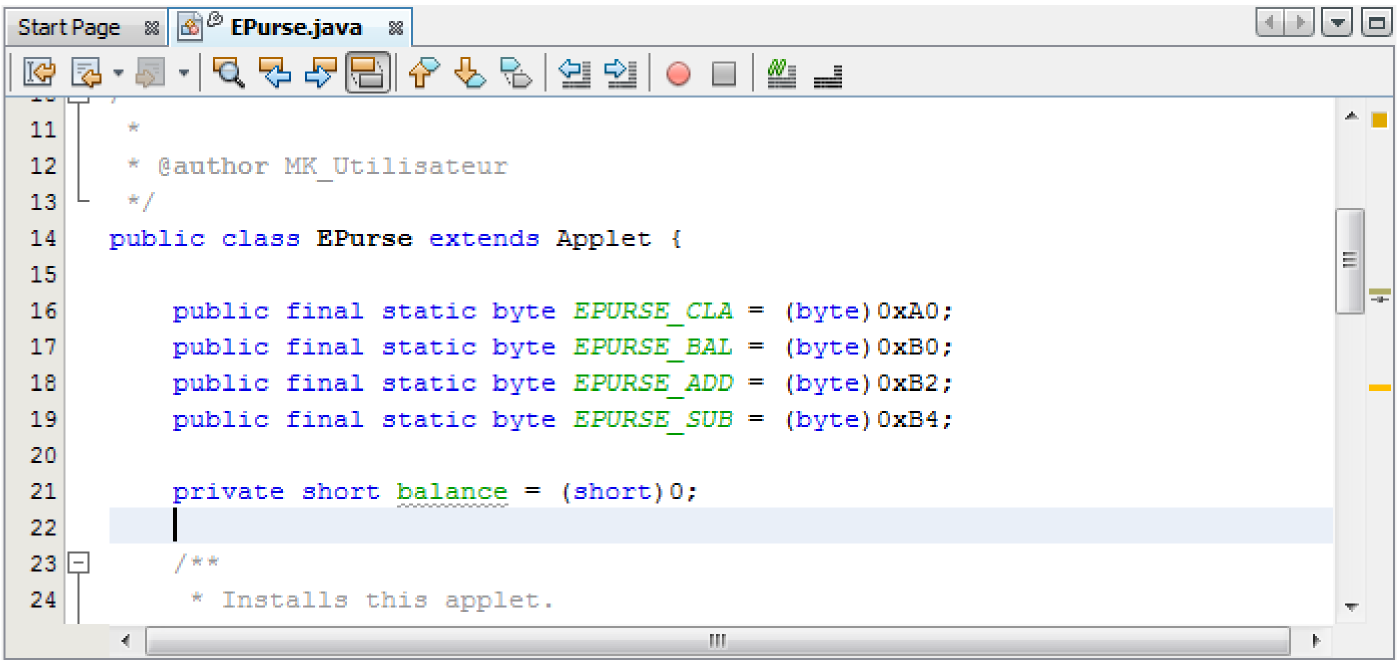
Click on **Finish**:



```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package epurse;

import javacard.framework.*;

/**
 *
 * @author MK_Utilisateur
 */
public class EPurse extends Applet {
```

The wizard had created the full project: the package **epurse** with a skeleton of the applet **EPurse.java**, the APDU scripts skeleton, and other important files which will be discussed later.

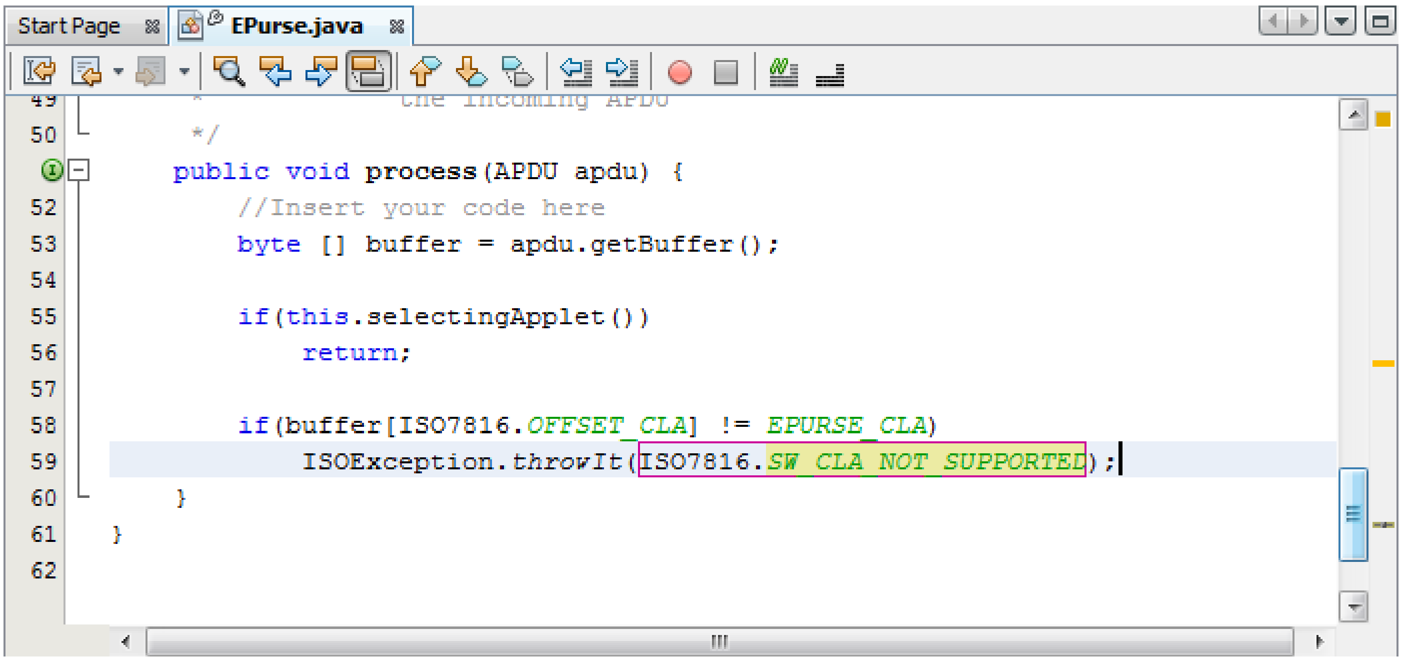
2. The **EPurse** applet supports three commands: get the balance of the e-purse, add some money to the e-purse, withdraw some money from the e-purse. Let's define four constants to handle the class identification and the three commands. Let's define also a field into the applet to handle the balance:



```
 *
 * @author MK_Utilisateur
 */
public class EPurse extends Applet {

    public final static byte EPURSE_CLA = (byte)0xA0;
    public final static byte EPURSE_BAL = (byte)0xB0;
    public final static byte EPURSE_ADD = (byte)0xB2;
    public final static byte EPURSE_SUB = (byte)0xB4;

    private short balance = (short)0;

    /**
     * Installs this applet.
```

Question: why it is necessary to cast the literal hex constants with (byte) and the value 0 with (short)?

3. Now let's implement the **process** method:



```
     */
    public void process(APDU apdu) {
        //Insert your code here
        byte [] buffer = apdu.getBuffer();

        if(this.selectingApplet())
            return;

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

First, we have to retrieve, from the **apdu**, the **buffer** in which we can find the APDU. Then we have to return without processing if the APDU is a command for ***selecting*** the ***applet***. Then we have also to exit with an error, if the CLA found in the APDU is not the expected one.

The Netbeans tool supports perfectly the Java Card environment. When you type in, for instance, **ISO7816.** , Netbeans proposes immediately the constants available in this "class".

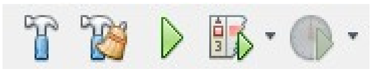
If everything is correct, we need to process the commands. The commands ADD and SUB need to exchange extra data with the CAD. We have to specify the amount of money to add to or to withdraw from the e-purse. This amount must be a short positive integer compatible with the existing balance in the e-purse. This short number is passed to the applet within two consecutive bytes in the APDU, in the CDATA area (starting with byte 5). The BAL command must return the balance of the e-purse within a short integer. This short integer is split down into the two first bytes of the return buffer. Here is the corresponding code:

```
short amount = (short)0;

switch(buffer[ISO7816.OFFSET_INS]){
   case EPURSE_BAL:
      Util.setShort(buffer, (short)0, balance);
      apdu.setOutgoingAndSend((short)0, (short)2);
      break;
   case EPURSE_ADD:
      apdu.setIncomingAndReceive();
      amount = Util.getShort(buffer, ISO7816.OFFSET_CDATA);
      if(amount <=0 || (short)(balance+amount) <= 0) // overloading
         ISOException.throwIt(ISO7816.SW_WRONG_DATA);
      else
         balance += amount;
      break;
   case EPURSE_SUB:
      apdu.setIncomingAndReceive();
      amount = Util.getShort(buffer, ISO7816.OFFSET_CDATA);
      if(amount <= 0 || balance < amount) // overloading
         ISOException.throwIt(ISO7816.SW_WRONG_DATA);
      else
         balance -= amount;
      break;
   default:
      ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
}
```

For ADD and SUB, amount is first tested to avoid negative or null amount of money for adding or withdrawing. Then amount is tested to check the validity of the future balance (to avoid exceeding the maximum amount allowed in the e-purse or to avoid withdrawing more money than the e-purse contains really...).

Question: why it is not necessary to cast with (short) the assignment += or -=?

3. Build the project by clicking on the hammer:



```
init-platform-properties:
Using JavaCard Platform Definition at C:\Users\MK_Utilisateur\.netbeans\6.9\config\Services\Platforms\org-netbeans-api-java-Platform\javacard_default.jcplatform
Java Card Home is C:\Program Files\NetBeans 6.9.1\javacard\JCDK3.0.2_ConnectedEdition (Java Card Platform)
init-ri-properties:
init-device-properties:
Platform device property name is jcplatform.javacard_default.devicespath
Computed device folder path is C:\Users\MK_Utilisateur\.netbeans\6.9\config\org-netbeans-modules-javacard\servers\javacard_default
Platform device file path property name is C:\Users\MK_Utilisateur\.netbeans\6.9\config\org-netbeans-modules-javacard\servers\javacard_default\Default Device.jcard
Deploying to device Default Device http port 8019
init-keystore:
Keystore is C:\Program Files\NetBeans 6.9.1\javacard\JCDK3.0.2_ConnectedEdition/samples/keystore/a.keystore
Created dir: C:\Temp\epurse\build
Created dir: C:\Temp\epurse\build\META-INF
Created dir: C:\Temp\epurse\build\APPLET-INF\classes
Created dir: C:\Temp\epurse\dist
build-dependencies:
unpack-dependencies:
compile:
Compiling 1 source file to C:\Temp\epurse\build\APPLET-INF\classes
compile-proxies:
create-descriptors:
Copying 1 file to C:\Temp\epurse\build\APPLET-INF
Copying 1 file to C:\Temp\epurse\build\META-INF
create-static-pages:
Copying 1 file to C:\Temp\epurse\build
do-pack:
Resolved dist.bundle as relative file C:\Temp\epurse\dist\EPurse.cap
Resolved build.dir as relative file C:\Temp\epurse\build
```

Netbeans creates directories in the project directory, particularly dist which will contain the CAP file

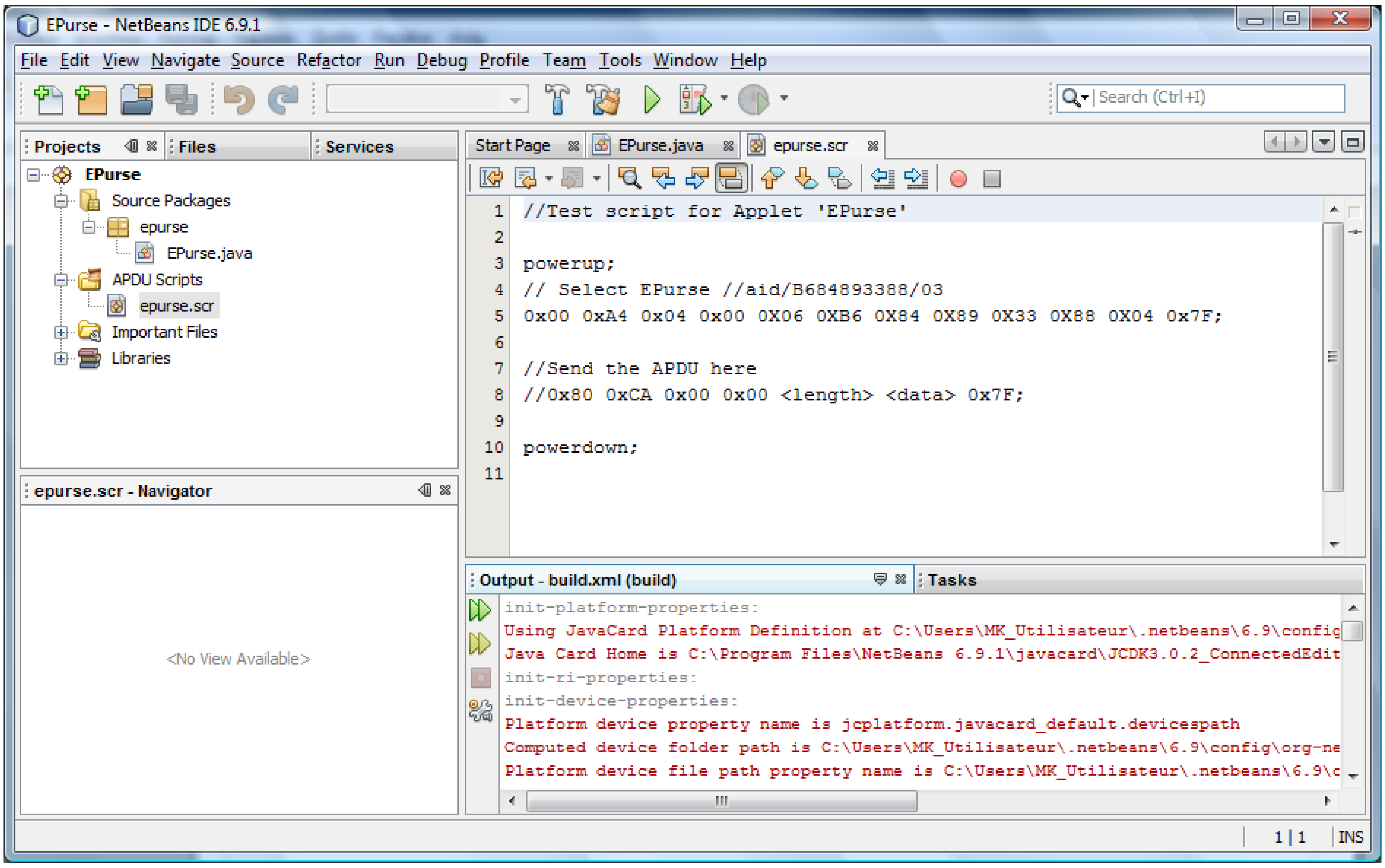
```
Packager [v3.0.2]
    Copyright (c) 2009 Sun Microsystems, Inc.
    All rights reserved.
    Use is subject to license terms.

[ INFO: ] Validating Command Line
[ INFO: ] Setting packaging information
[ INFO: ] Package is being created
[ INFO: ] Converter [v3.0.2]
[ INFO: ]     Copyright (c) 2009 Sun Microsystems, Inc.
    All rights reserved.
    Use is subject to license terms.
[ INFO: ] conversion completed with 0 errors and 0 warnings.
[ INFO: ] Printing diagnostics..
```

The converter converts the byte code produced by the compiler into a code which can be loaded into the Java Card

```
[ INFO: ] Packager/create SUCCESS [0 error(s) and 0 warning(s)]
pack:
do-sign:
sign:
build:
BUILD SUCCESSFUL (total time: 6 seconds)
```

4. The next operation is the preparation of the script for the simulation. In order to test our applet, we have to prepare APDUs which could be sent to the applet for processing. The simulator consists into two processes: the first one simulates the Java Card with the corresponding applet, the second simulates the Card Acceptance Device. The two processes communicate using a socket according to a protocol which is called TLP224 (a very old card reader from Bull CP8). The Java Card project wizard had prepared a skeleton for the script.

In the project window, open the **APDU scripts** folder, and double-click on the **epurse.scr** script to open the corresponding file for edition:



```
//Test script for Applet 'EPurse'

powerup;
// Select EPurse //aid/B684893388/03
0x00 0xA4 0x04 0x00 0X06 0XB6 0X84 0X89 0X33 0X88 0X04 0x7F;

//Send the APDU here
//0x80 0xCA 0x00 0x00 <length> <data> 0x7F;

powerdown;
```

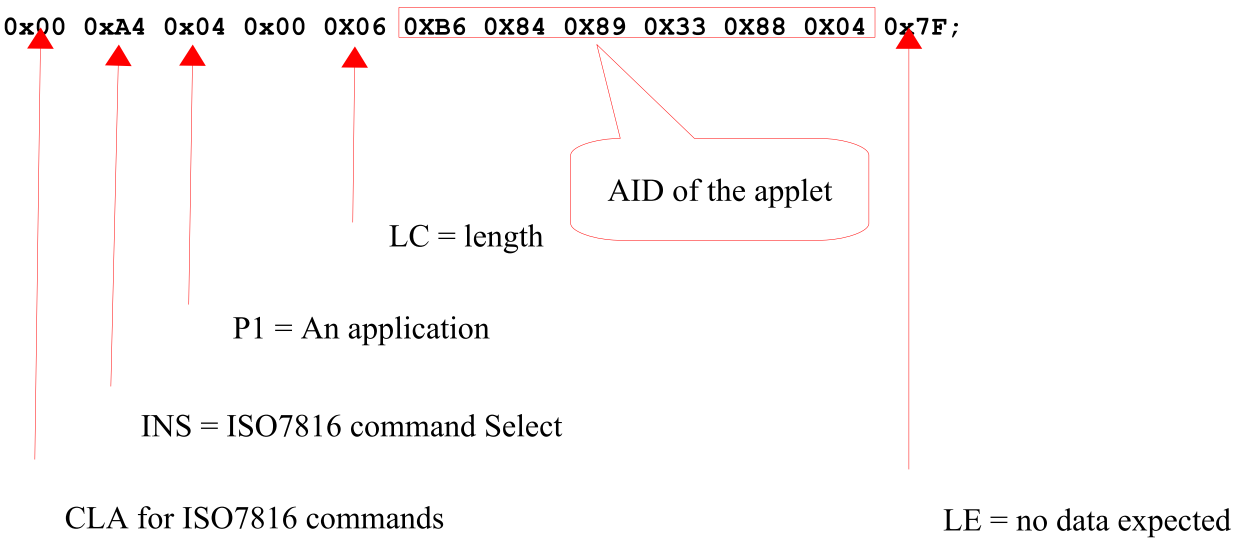
```
init-platform-properties:
Using JavaCard Platform Definition at C:\Users\MK_Utilisateur\.netbeans\6.9\config
Java Card Home is C:\Program Files\NetBeans 6.9.1\javacard\JCDK3.0.2_ConnectedEdit
init-ri-properties:
init-device-properties:
Platform device property name is jcplatform.javacard_default.devicespath
Computed device folder path is C:\Users\MK_Utilisateur\.netbeans\6.9\config\org-ne
Platform device file path property name is C:\Users\MK_Utilisateur\.netbeans\6.9\c
```

This is the skeleton already generated by the Java Card project wizard. The simulator uses a scripting language very close to the APDU which must be sent to the Java Card.

Comments appear in line starting with a double slash (//) like in Java. All the statements are terminated by a semi-colon (**;**). The first statement is the **powerup** in order to start the Java Card itself. The last statement is the **powerdown** in order to stop the Java Card. This simulator is not a real simulator because, at the end of the simulation, the simulated Java Card is destroyed, and cannot keep changes in memory. But this simulator is enough to test the behavior of the applet.

The statements are very close to the APDU: they contain, separated by spaces, the integer values which constitute the APDU and the data to be sent to the applet. Integer value could be written in decimal or in hexadecimal (provide to prefix the number with 0x like in Java).

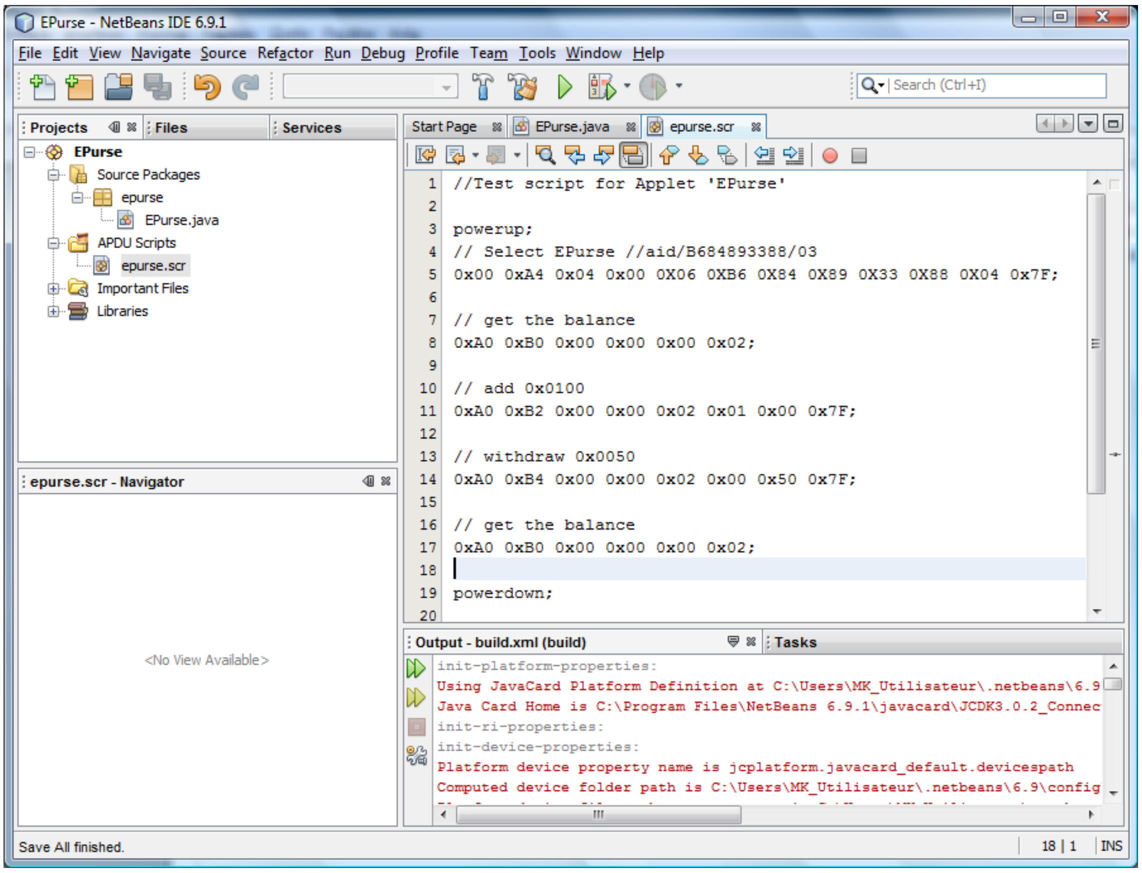
The simulation supposes that the applet is already installed into the Java Card. The first APDU sent after the **powerup** is for selecting the applet:



The scripting language of the simulator doesn't know if your APDU sends data to the Java Card or expects data from the Java Card. We have to specify the values of **LC** AND **LE** in the same APDU. For LE, we can specify the number of bytes expected. If no data are expected from the Java Card, the value of LE must be **0x7F** (TLP224 protocol...).

Edit the **epurse.scr** script file in order to do the following operations:

- get the balance of the e-purse (normally 0x0000)
- add 0x0100 to the e-purse
- withdraw 0x0050 from the e-purse
- get the balance of the e-purse (normally 0x00B0)

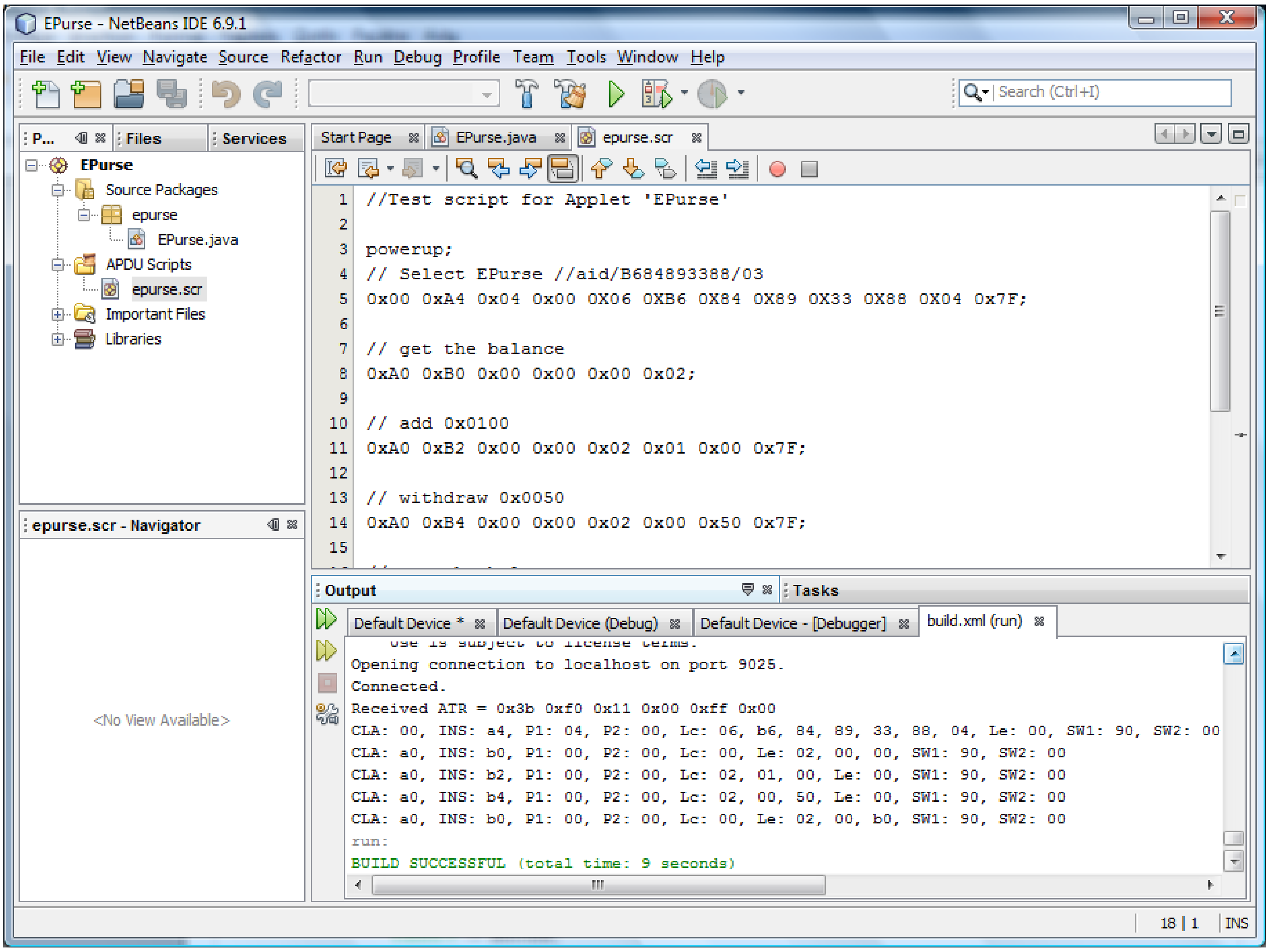


```
//Test script for Applet 'EPurse'

powerup;
// Select EPurse //aid/B684893388/03
0x00 0xA4 0x04 0x00 0X06 0XB6 0X84 0X89 0X33 0X88 0X04 0x7F;

// get the balance
0xA0 0xB0 0x00 0x00 0x00 0x02;

// add 0x0100
0xA0 0xB2 0x00 0x00 0x02 0x01 0x00 0x7F;

// withdraw 0x0050
0xA0 0xB4 0x00 0x00 0x02 0x00 0x50 0x7F;

// get the balance
0xA0 0xB0 0x00 0x00 0x00 0x02;

powerdown;
```

```
init-platform-properties:
Using JavaCard Platform Definition at C:\Users\MK_Utilisateur\.netbeans\6.9
Java Card Home is C:\Program Files\NetBeans 6.9.1\javacard\JCDK3.0.2_Connec
init-ri-properties:
init-device-properties:
Platform device property name is jcplatform.javacard_default.devicespath
Computed device folder path is C:\Users\MK_Utilisateur\.netbeans\6.9\config
```

Question: why the statements "**get the balance**" contain six bytes? Describe them.

5. Save the script. We are ready to run the simulator. Click on the green arrow into the toolbar. After few seconds (time needed to launch the two processes, install the applet and initialize the socket...), the script is run and we can get the result


EPurse - NetBeans IDE 6.9.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
P... Files Services
EPurse
Source Packages
epurse
EPurse.java
APDU Scripts
epurse.scr
Important Files
Libraries
epurse.scr - Navigator
<No View Available>
Start Page EPurse.java epurse.scr
Output Tasks
Default Device * Default Device (Debug) Default Device - [Debugger] build.xml (run)
18 | 1 INS

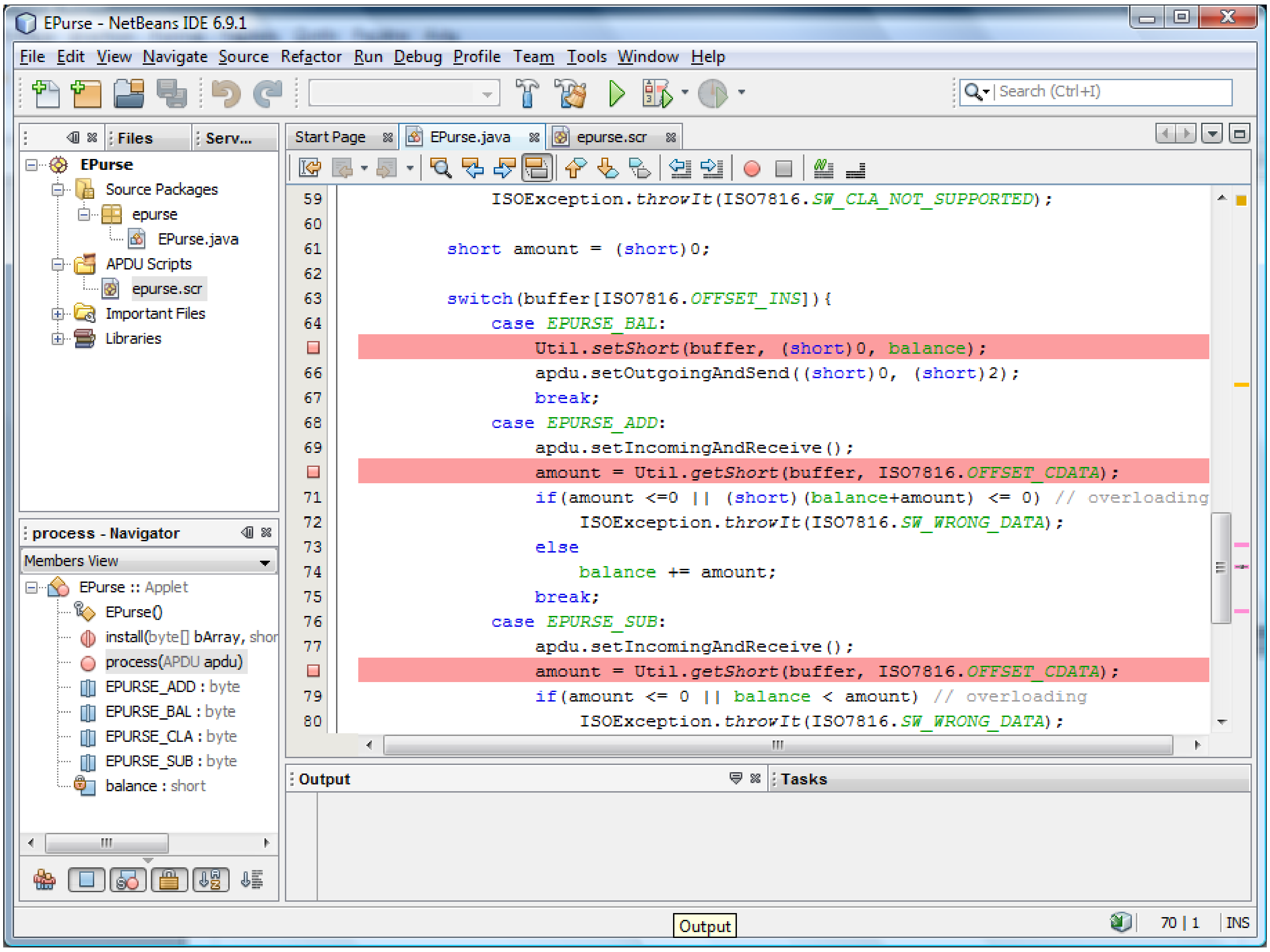
```
//Test script for Applet 'EPurse'

powerup;
// Select EPurse //aid/B684893388/03
0x00 0xA4 0x04 0x00 0X06 0XB6 0X84 0X89 0X33 0X88 0X04 0x7F;

// get the balance
0xA0 0xB0 0x00 0x00 0x00 0x02;

// add 0x0100
0xA0 0xB2 0x00 0x00 0x02 0x01 0x00 0x7F;

// withdraw 0x0050
0xA0 0xB4 0x00 0x00 0x02 0x00 0x50 0x7F;
```

```
use is subject to license terms.
Opening connection to localhost on port 9025.
Connected.
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 06, b6, 84, 89, 33, 88, 04, Le: 00, SW1: 90, SW2: 00
CLA: a0, INS: b0, P1: 00, P2: 00, Lc: 00, Le: 02, 00, 00, SW1: 90, SW2: 00
CLA: a0, INS: b2, P1: 00, P2: 00, Lc: 02, 01, 00, Le: 00, SW1: 90, SW2: 00
CLA: a0, INS: b4, P1: 00, P2: 00, Lc: 02, 00, 50, Le: 00, SW1: 90, SW2: 00
CLA: a0, INS: b0, P1: 00, P2: 00, Lc: 00, Le: 02, 00, b0, SW1: 90, SW2: 00
run:
BUILD SUCCESSFUL (total time: 9 seconds)
```

Questions: what does mean "`SW1: 90, SW2: 00`"? What does mean "`ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00`"? Have you found the value of the balance after the second "**get balance**"?

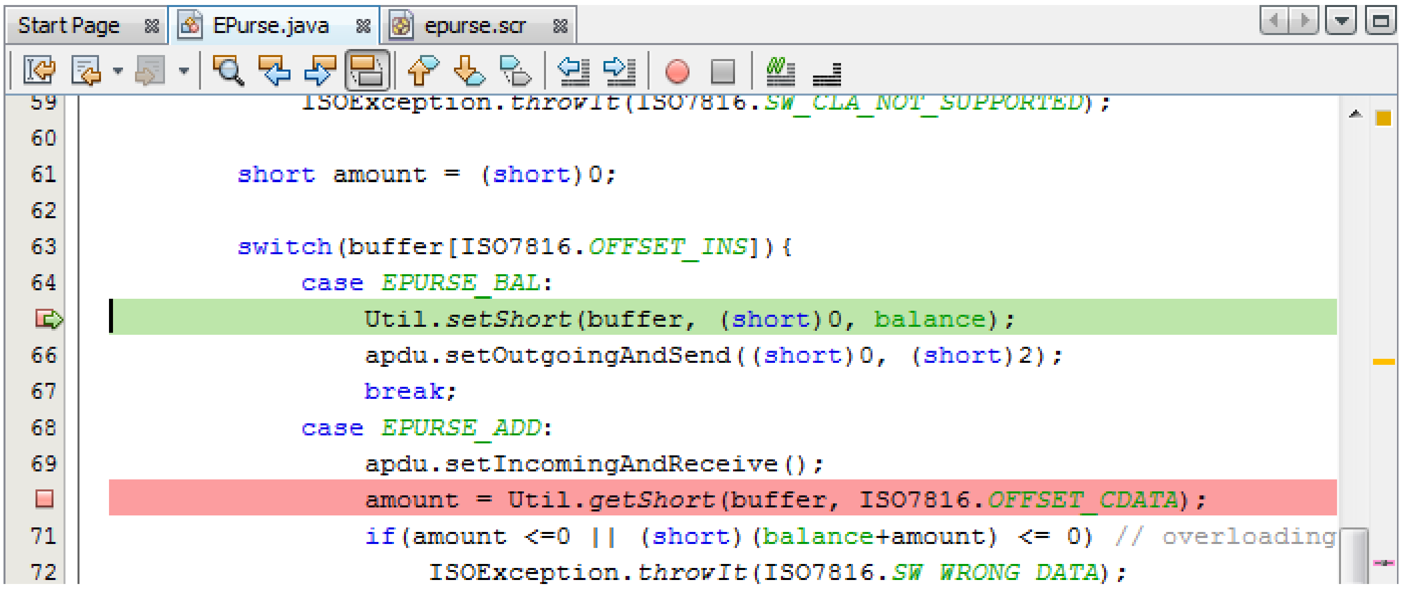
6. One of the most interesting feature of an Integrated Development Environment like Netbeans is the debugging tool. It is possible to stop the running of the applet where you want and get the value of local variables at a finger snap. This feature is impossible when the applet is uploaded into a real Java Card, for security reasons.

To setup a stop in the applet, what is called a break point, just click on the statement number facing the statement where you want to stop. Next screen capture shows the result of the setting of three break points at the beginning of the process of the three commands: BALance, ADDing money, SUBstracting money.



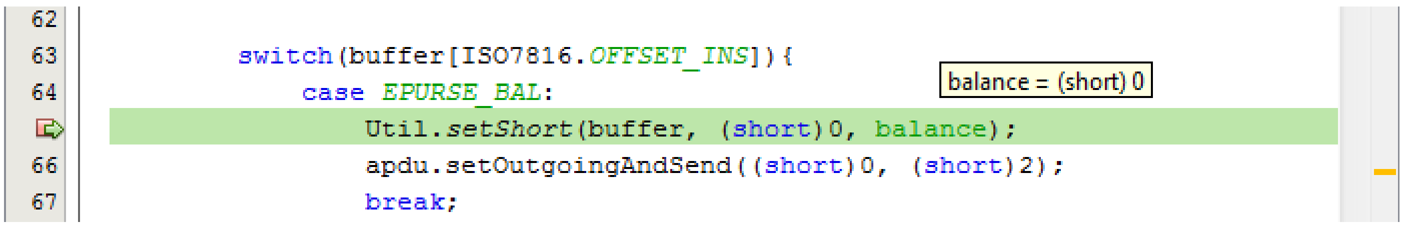
```
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

        short amount = (short)0;

        switch(buffer[ISO7816.OFFSET_INS]){
            case EPURSE_BAL:
                Util.setShort(buffer, (short)0, balance);
                apdu.setOutgoingAndSend((short)0, (short)2);
                break;
            case EPURSE_ADD:
                apdu.setIncomingAndReceive();
                amount = Util.getShort(buffer, ISO7816.OFFSET_CDATA);
                if(amount <=0 || (short)(balance+amount) <= 0) // overloading
                    ISOException.throwIt(ISO7816.SW_WRONG_DATA);
                else
                    balance += amount;
                break;
            case EPURSE_SUB:
                apdu.setIncomingAndReceive();
                amount = Util.getShort(buffer, ISO7816.OFFSET_CDATA);
                if(amount <= 0 || balance < amount) // overloading
                    ISOException.throwIt(ISO7816.SW_WRONG_DATA);
```

The project must be launched using the menu **Debug | Debug Main Project**. The applet stops on the first break point:



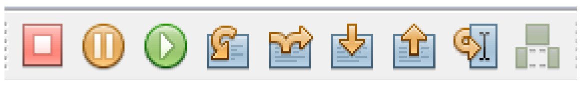
```
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

        short amount = (short)0;

        switch(buffer[ISO7816.OFFSET_INS]){
            case EPURSE_BAL:
                Util.setShort(buffer, (short)0, balance);
                apdu.setOutgoingAndSend((short)0, (short)2);
                break;
            case EPURSE_ADD:
                apdu.setIncomingAndReceive();
                amount = Util.getShort(buffer, ISO7816.OFFSET_CDATA);
                if(amount <=0 || (short)(balance+amount) <= 0) // overloading
                    ISOException.throwIt(ISO7816.SW_WRONG_DATA);
```

To discover the value of the local variables, you can hoover these variables directly on the source code:

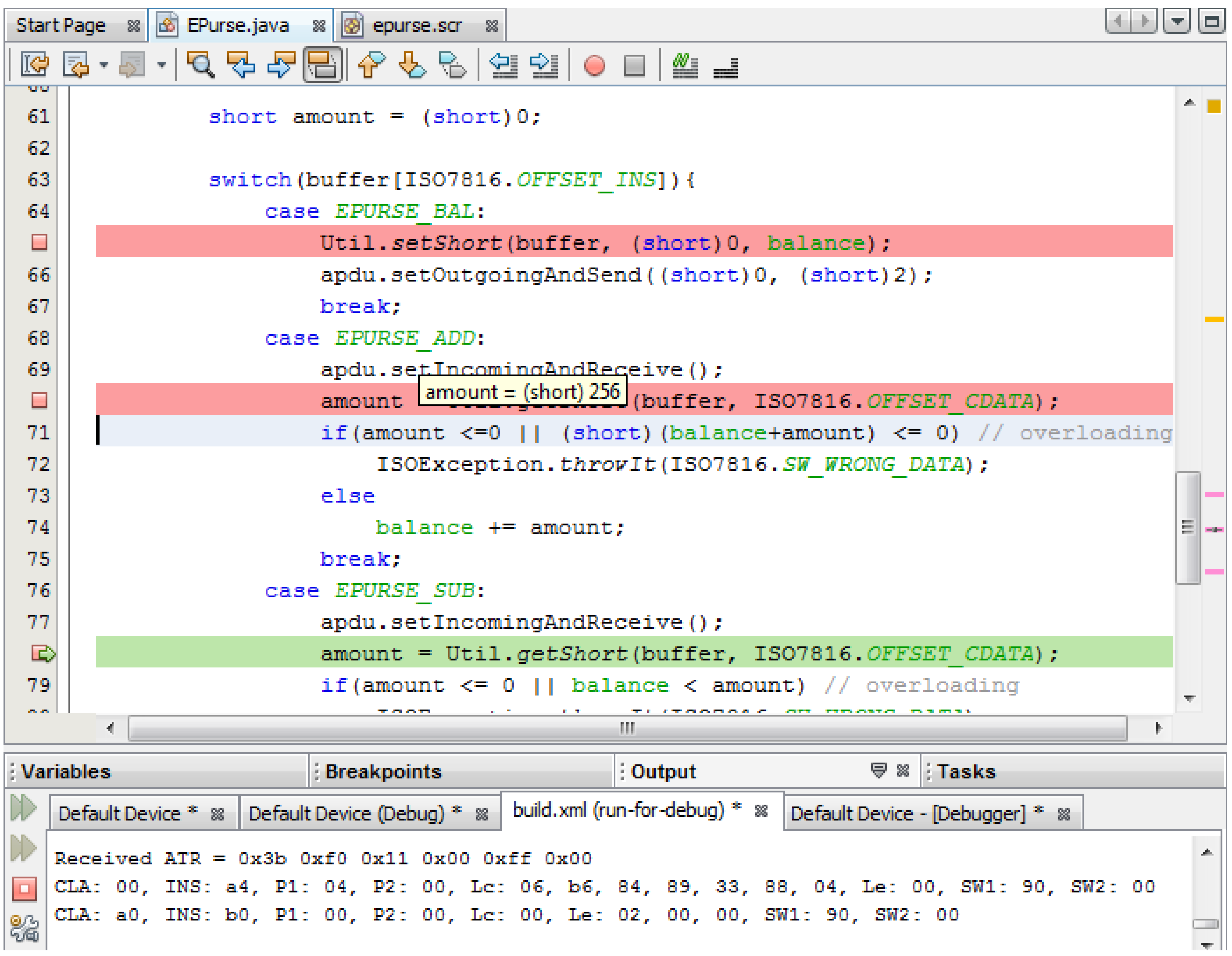


```
        switch(buffer[ISO7816.OFFSET_INS]){
            case EPURSE_BAL:
                Util.setShort(buffer, (short)0, balance);
                apdu.setOutgoingAndSend((short)0, (short)2);
                break;
```

You can step through the applet using one of the buttons of the toolbar:



The first arrow means **Step over (F8)**, the second arrow means **Step over expression (Maj+F8)**, the third arrow means **Step into (F7)**, the fourth arrow means **Step out (Ctrl+F7)**, the fifth arrow means **Run to cursor (F4)**.



```
        short amount = (short)0;

        switch(buffer[ISO7816.OFFSET_INS]){
            case EPURSE_BAL:
                Util.setShort(buffer, (short)0, balance);
                apdu.setOutgoingAndSend((short)0, (short)2);
                break;
            case EPURSE_ADD:
                apdu.setIncomingAndReceive();
                amount = Util.getShort(buffer, ISO7816.OFFSET_CDATA);
                if(amount <=0 || (short)(balance+amount) <= 0) // overloading
                    ISOException.throwIt(ISO7816.SW_WRONG_DATA);
                else
                    balance += amount;
                break;
            case EPURSE_SUB:
                apdu.setIncomingAndReceive();
                amount = Util.getShort(buffer, ISO7816.OFFSET_CDATA);
                if(amount <= 0 || balance < amount) // overloading
```

```
Received ATR = 0x3b 0xf0 0x11 0x00 0xff 0x00
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 06, b6, 84, 89, 33, 88, 04, Le: 00, SW1: 90, SW2: 00
CLA: a0, INS: b0, P1: 00, P2: 00, Lc: 00, Le: 02, 00, 00, SW1: 90, SW2: 00
```

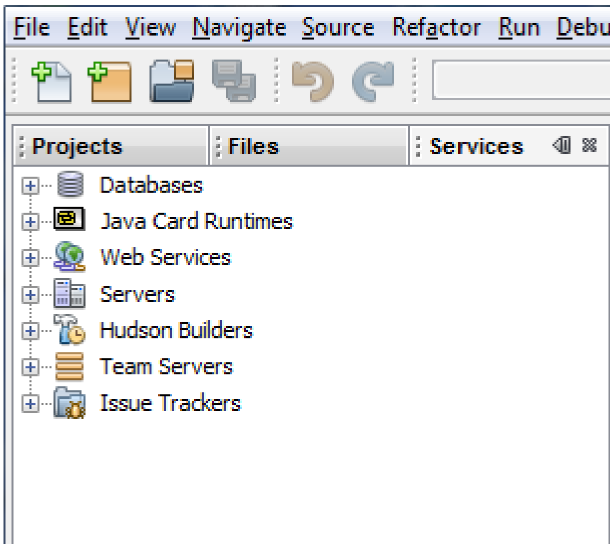
This screen capture was obtained after some **Step over**.

7. **Hands-on**: it is time to test all the branches of our applet. Modify the script in order to check the behavior of the applet when we try to add or subtract a negative number, or to add a number which could exceed the maximum value of the balance, or to subtract a number larger than the balance.

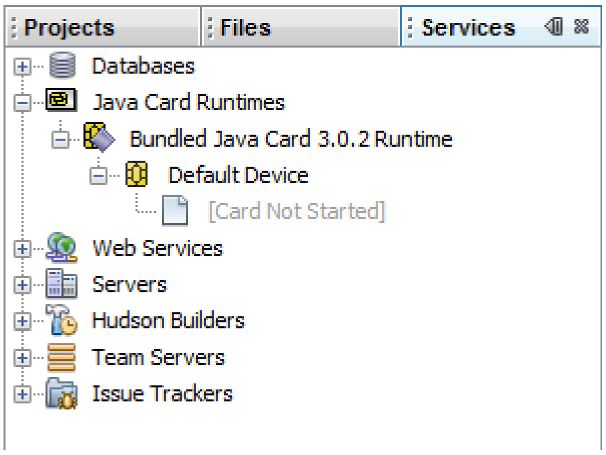
8. The simulation with a static script is not convenient when a real dialog between the applet and the CAD is necessary. For instance, if the CAD doesn't know the size of the data expected from an APDU, it needs to ask the applet about this size before sending the APDU with the correct LE parameter.

Netbeans proposes a solution to solve this problem through an interactive tool which is called **the console**. The process to run the applet is slightly different. The simulation is closer to reality: the Java Card is started first, then the applet is uploaded and installed, then, through the console, it is possible to send real APDU to the applet.

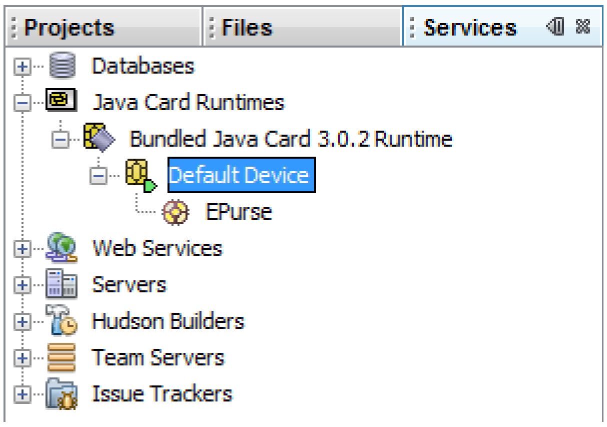
On the project window, select the **Services** tab:



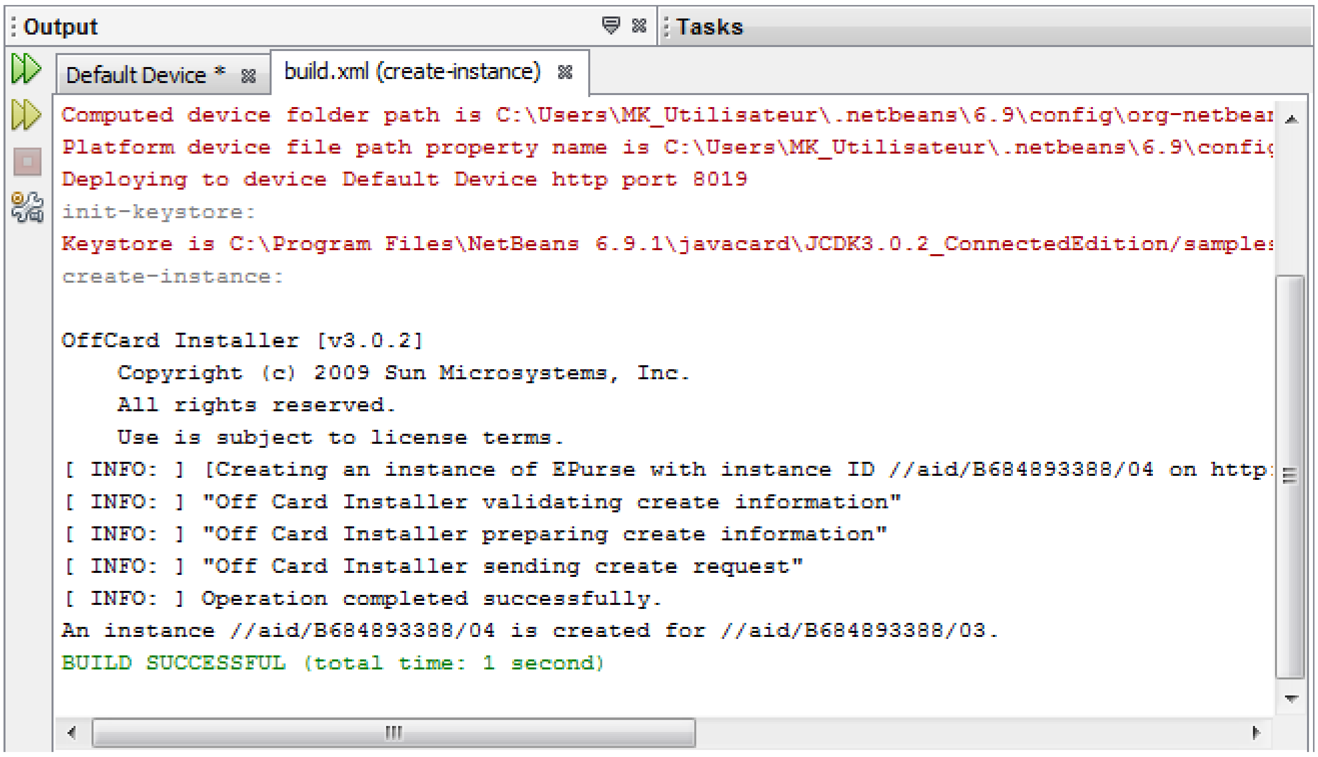
Develop the branch **Java Card Runtimes | Bundled Java Card 3.0.2 Runtime | Default Device** :



No card is started for the moment. Right click on **Default Device** and select **Start**, which starts the default Java Card. The **Output** window must display **card ready**. Select the **Projects** tab and in the window selected, right click on the project **EPurse.** Select the item **(Re)Load**. The CAP file is uploaded into the Java Card. This can be confirmed by looking at the window **Services**:

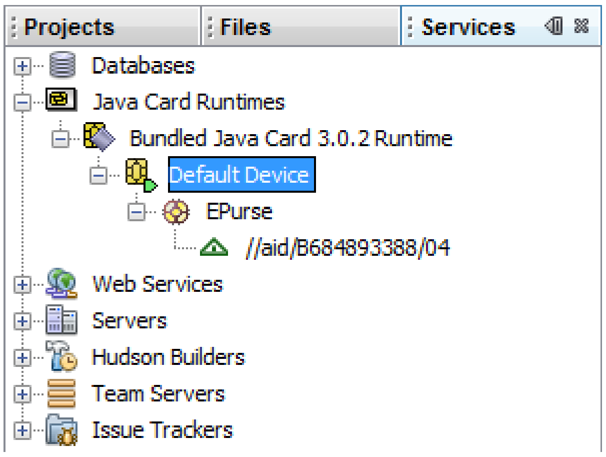


Return to the window **Projects** and right click on the project **EPurse** and select **Create Instance(s)**. The **install** method is run and the instance of the **EPurse** class is created and associated with **aid**. The **Outputs** window shows the creation of the instance of the **EPurse**:



```
Computed device folder path is C:\Users\MK_Utilisateur\.netbeans\6.9\config\org-netbear
Platform device file path property name is C:\Users\MK_Utilisateur\.netbeans\6.9\config
Deploying to device Default Device http port 8019
init-keystore:
Keystore is C:\Program Files\NetBeans 6.9.1\javacard\JCDK3.0.2_ConnectedEdition/samples
create-instance:

OffCard Installer [v3.0.2]
    Copyright (c) 2009 Sun Microsystems, Inc.
    All rights reserved.
    Use is subject to license terms.
[ INFO: ] [Creating an instance of EPurse with instance ID //aid/B684893388/04 on http:
[ INFO: ] "Off Card Installer validating create information"
[ INFO: ] "Off Card Installer preparing create information"
[ INFO: ] "Off Card Installer sending create request"
[ INFO: ] Operation completed successfully.
An instance //aid/B684893388/04 is created for //aid/B684893388/03.
BUILD SUCCESSFUL (total time: 1 second)
```

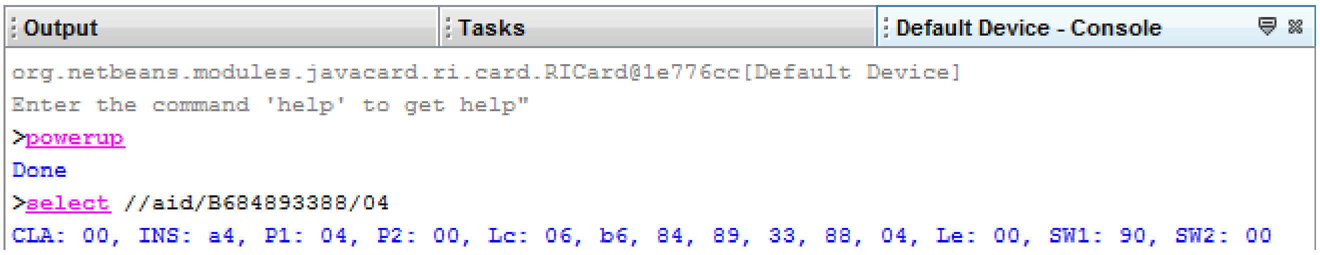
This is confirmed in the **Services** window:



Right click on **Default Device** and select **Open Console**. It is possible, through this console to send APDU directly to the applet which has be uploaded into the Java Card. To power up and select the applet, type in the console window:

```
powerup
select //aid/B684893388/04
```

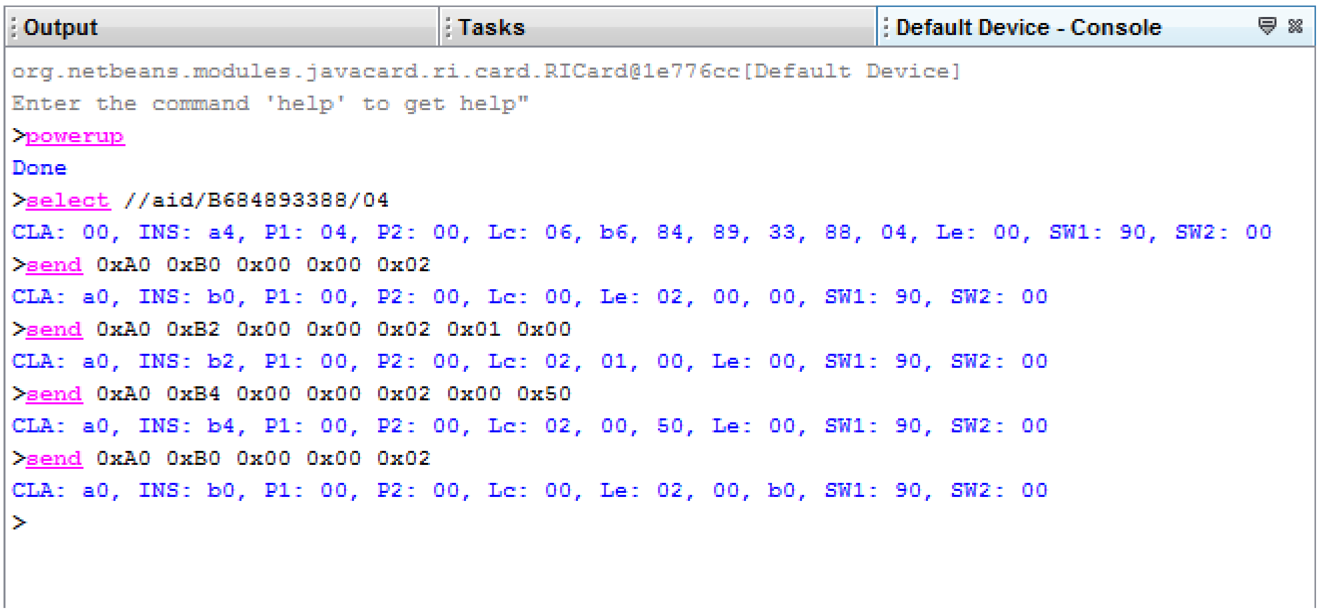
Probably you have to change the value of the AID which could be different on your PC (take a copy from the script):



Output | Tasks | Default Device - Console

```
org.netbeans.modules.javacard.ri.card.RICard@1e776cc[Default Device]
Enter the command 'help' to get help"
>powerup
Done
>select //aid/B684893388/04
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 06, b6, 84, 89, 33, 88, 04, Le: 00, SW1: 90, SW2: 00
```

To send an APDU, you have just to type in the keyword **send**, followed by the APDU written as spaced hex numbers. You have to type in the real APDUs and not the statements used in the scripts (which are compliant with the TLP224):



Output | Tasks | Default Device - Console

```
org.netbeans.modules.javacard.ri.card.RICard@1e776cc[Default Device]
Enter the command 'help' to get help"
>powerup
Done
>select //aid/B684893388/04
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 06, b6, 84, 89, 33, 88, 04, Le: 00, SW1: 90, SW2: 00
>send 0xA0 0xB0 0x00 0x00 0x02
CLA: a0, INS: b0, P1: 00, P2: 00, Lc: 00, Le: 02, 00, 00, SW1: 90, SW2: 00
>send 0xA0 0xB2 0x00 0x00 0x02 0x01 0x00
CLA: a0, INS: b2, P1: 00, P2: 00, Lc: 02, 01, 00, Le: 00, SW1: 90, SW2: 00
>send 0xA0 0xB4 0x00 0x00 0x02 0x00 0x50
CLA: a0, INS: b4, P1: 00, P2: 00, Lc: 02, 00, 50, Le: 00, SW1: 90, SW2: 00
>send 0xA0 0xB0 0x00 0x00 0x02
CLA: a0, INS: b0, P1: 00, P2: 00, Lc: 00, Le: 02, 00, b0, SW1: 90, SW2: 00
>
```

It is possible to **powerdown** the Java Card, then **powerup** it , **select** back the applet, and get the balance: the simulator keeps in memory the status of the card when it is **powerdown** and is capable to retrieve it at the **powerup**.

9. **Hands-on**: test the applet using the APDUs to check all the branches of the applet.

10. **Hands-on**: add a **OwnerPIN** to your applet. Add two extra commands: the first one to check the **OwnerPIN**, the second to modify the **OwnerPIN**.