# Contents

# Basic Java Card Development

Northern Arizona University

INF633B, Spring 2018, Dr. Bertrand Cambou

Written By: Christopher Robert Philabaum <cp723@nau.edu>

## Preparation

### Install Java 8 JRE (32-bit)

This is necessarily so that the Java Card IDE can run, as it requires a 32-bit install of Java

1. Go to http://www.oracle.com/technetwork/java/javase/downloads/index.html
2. Scroll down to **Java SE 8uXXX** (or **Java SE 8uXXX / 8uYYY** when there are two related patches)
3. Click on **Download** under *JRE* to the right.
4. Go to the first patch (**8uXXX**).
5. Accept the license agreement.
6. Click on the download for *Windows x86 Offline* and run the executable.

## Install Java SDK 8

This SDK is the more general deveopment kit for Java that is needed by IntelliJ.

1. Go to http://www.oracle.com/technetwork/java/javase/downloads/index.html
2. Scroll down to **Java SE 8uXXX** (or **Java SE 8uXXX / 8uYYY** when there are two related patches)
3. Click on **Download** under *JDK* to the right.
4. Go to the first patch (**8uXXX**).
5. Accept the license agreement.
6. Click on the download for *Windows x64* (assuming that you're on a 64-bit system) and run the executable.

   *Note: The SDK executable includes the JRE runtime for the same version, so you don't need to install the JRE separately.*

## Install Java Card IDE

Java Card IDE is a proprietary skin over Eclipse that by default includes the tools and development kits for Java Card development.

1. Go to your Java Card USB stick's path in File Explorer.
2. Go to `Infineon/JC IDE/`
3. Install `JavaCard-IDE-2.5.exe` on to your computer.

## Install JetBrains IntelliJ IDEA Community Edition

JetBrains IntelliJ IDEA Community Edition is a Java IDE that we're using for the terminal application.

1. Go to https://www.jetbrains.com/toolbox/app/
2. Download the `.exe` and run the installer.
3. Optional: Sign In to a JetBrains account once JetBrains Toolbox opens, or hit `Skip` on the bottom-right corner.
4. Under JetBrains Toolbox, install IntelliJ IDEA Community

   *Note: The difference between* Community* and *Ultimate* is that *Community* is the open source version and is free to use in education and research context (while having more limited features).

## Java Card Applets

### Things to Know Before Creating an Applet

### API

For a reference to the Java Card v2.2.2 API online, look to here.

### APDU Command / Response Structure

Refer to here to familiarize yourself on with the APDU commands and responses that smart cards use to communicate with a terminal application.

### APDU Status Words

Sometimes things don't always go as planned and you'll receive an error message from the smart card. Look to here for a general list on the different semantics on what each status words means.

The most common you'll run into are:

- **90 00:** Success. All is good and nothing (exceptional) went wrong. This doesn't mean your code isn't wrong, but at least it didn't do something that would have caused a crash or accessed something it wasn't allowed to.
- **6F 00:** A generic internal error, e.g. memory access violation.
- **6D 00:** The instruction isn't supported (as defined by you or if you forgot to select the applet first).
- **6A 82:** File not found. Often found when trying to select an applet that's not there. You'll run into this most often from a failed installation.
- **67 00:** Wrong data length used. This will tend to occurr when you expect a specfic range of lengths and throw the error back when the length is too big or too small.

### JCDK Quirks

- Like Java, all integral data types (`byte`, `short`, `int`, `long`) are signed and have no unsigned alternatives.
- Unlike Java, you are only limited to 16-bit word sizes, aka. only `byte` & `short`.
- Combining the two above means that overflow (and underflow) are more common problems to consider when doing multiplication and modular arithmetic.
- `String` types and related functions don't exist, so any and all text-like data should be represented by raw `byte` arrays.

- Garbage collection is ***not*** a given and depends on a card-by-card basis. Don't assume that instantiated objects will be removed automatically (aka. `byte` or `short` arrays, see below).
- You have a very limited amount of resources in both RAM and EEPROM, so be mindful of not allocating more than a dozen of kilobytes of memory at a time.

### Debugging (in Java Card IDE)

- Because of the lack of plugins tied specifically to the cards we have, the JavaCard IDE cannot stop on breakpoints in debug mode. In order to debug it, you'll have to rely on indirect, creative ways such as throwing an exception at a specific point if you want to check if the code safely ran that far (aka. via `ISOException.throwIt(randomNonUsedStatusWordHere);`)

### Memory

- Any array created with the `new` keyword like `this.array = new byte[25];` is stored on *permanent*, non-volatile storage, most typically the EEPROM.

  > *Never use the* ***`new`*** *keyword in any block of code but the constructor's.* Aka. if our applet is called `EchoApplet` then our constructor will look like `private KVL (byte[] bArray, short bOffset, byte bLength)`.
  >
  > If you don't adhere to this, your card will eventually run out of permanent storage and throw an obscure error when it tries to allocate more ] in the future.

- To create an array in volatile memory, use the `JCSystem.makeTransientByteArray()` function.
- Make sure to declare any arrays as *class variables* instead of local variables. For example, declaring `private short[] values;` vs. using `short[] values;` within a function.
- By extension, only initialize these arrays within the constructor, which allocates the memory when you first install the applet on the card.
- Watch out for any other sort of object instantiation and make sure those two are instantiated in the constructor. E.g., Where `private MessageDigest digestSHA;` is a class variable, and you instantiate it in the constructor as:

  `digestSHA = MessageDigest.getInstance(MessageDigest.ALG_SHA_512, false);`.

### Creating a Java Card Project

1. Open Java Card IDE.

2. Select the workspace to a folder you have access to, like:

   `C:\Users\$usernameHere$\Documents\JCProjects\`

   You might have problems using the default workspace path since that points to where Java Card IDE was installed.

3. Go to `File > New > Java Card Project`.

   > **Project Name:** Set the name of your project here. In my case, I used `EchoApplet`.

4. By default, Java Card IDE has a development environment labeled `custom` that assumes support for JCDK 3.0.1. However, the cards you have only support JCDK 2.2.2. To remedy this, click on `Add Development Environments`

5. This will open a new window. From here, select `New...`.

   > **Name:** We'll name it after the chipset, which is `NXP JA020`. **JCDK Converter:** Select `JCDK 2.2.2` **Java Compiler compliance:** Since JCDK 2.2.2 was based on Java 1.5, compliance, set it to this.

6. Click `Add...`

7. Select "Global Platform 2.1.1v1.0", and hit the `>` button to select it, then press `OK`.

8. Click `OK` to finish adding the development environment.

### Creating a Java Card Package and Applet

1. Go to `File > New > Java Card Package`.

2. **Name:** Java packages are usually written as the reverse uri of your organization. e.g. `www.google.com` becomes `com.google.www`. In our case, we'll use `edu.nau.siccs`.

   **AID:** The AID identifies the package (and the applet) and must be at least 5 bytes. It's written out as hex character. Since we're not worrying about having multiple other packages on the same card, we'll use something simple like: `01 02 03 04 05`

3. Leave everything else as is and then hit `Next >`.

4. **Applet:** This is the name of the applet and should be treated like a class name in Java. Be sure to set in the applet name in camel case `LikeThis`. For the demo we'll use `EchoApplet`

   **Applet AID:** The applet AID builds on the package AID and adds additional hex bytes to differentiate one applet from another. Usually I just count up from 1, so the first applet AID would be `01 02 03 04 05 01`, then `01 02 03 04 05 02`, and so on.

5. Now hit `Finish` to create your package and applet.
6. If you want to create any other applets under the same package, next time use `File > New > Java Card Applet` to create new applets under the same package.
7. For the demo, copy the source code from the `EchoApplet.java` file and replace the code you currently have.

## Installing a Java Card Applet on a Compatible Smart Card

### Setup Run Configuration

1. Go to `Run > Run Configurations`
2. Select `IFX Java Card` then hit the top-left `New` icon to create a new configration.
3. Name the configuration something sensible.
4. Have `Run Target` selected to `Sample Card`.
5. `Security Domain for Content Management` depends on how the card is configured. For the cards we have, I've found `OP Issuer SD` to work.
6. Click `Apply` then close the window.

### Install Java Card Applet

1. Insert the reader into the computer if you haven't already.

   *Note: If the reader was plugged in after Jva Card IDE has been opened, then you'll have to update the reader list to recognize it. At the top of the screen with the drop-down list that includes "Infineon Virtual Card Reader", press the "refresh" button to the left of it.*

2. Select the appripriate reader that the card is connected to (from the dropdown list mentioned aboe), in this case something like "HID Global OMNIKEY 3x21 Smart Card Reader 0"
3. To the right of the dropdown list, press the *Connect to card reader.*
4. Further to the right, select the arrow to the right of the "Play" button, then select the configratuion from before.

   If you get an error such as `6A82`, make sure the `Security Domain for Content Management` is set to `OP Issuer SD` (or `GP Issuer SD` depending on the card).

5. You'll see a progress bar on the bottom right. You'll know it is done installing once it disappears.

### Debugging a Java Card Applet

1. By default, Java Card IDE is set to `JC Development IFX` which changes the UI configration. At the top-right, press the button to the left of the text, and select `JC Debug (IFX)`.
2. Now the UI changes slightly, where on the right you should see a `Communicator` tab opened. Press `Manage...`.
3. Press `Add...` to create a new "Command Set".

   > **Command Set Type:** Leave this set to `Application`.
   >
   > **Command Set Name:** Set this related to your applet name, I just did "Echo". **AID:** This should be set to the same hex bytes as what your Applet AID is set to. In my case it's `01 02 03 04 05 01`. Make sure it's the longer Applet AID and not just the Package AID.

4. Once this is done hit `OK` and close out the *Manage Command Sets* window. On the dropdown menu within the `Communicator` tab, select the new command set you created `Echo`.
5. Now that you have the applet's command set set, with the card connected from before, hit `Select`.

   > *Note: You must make sure the applet is selected after you've connected to the card. If you don't do this, you'll get an obscure error such as `69 A5` when you try to send your own instructions since it's trying to connect the the Conte Management's instructions.

6. Under Command APDU, we'll write our first instruction. In the demo, the first instruction, `INS_ECHO`, has the `INS_BYTE` value set to `0x01`.

   Type out this in the field: `80 01 00 00 04 01 02 03 04`

   While you should read up on APDU command structures with the Wikipedia article, here's a long-short of what's going on here.

   **80 (CLA):** This is the Class byte and has some semantic meanings based on the individual bits set. Basically `8X` is not reserved for anything so we'll just use that.

   **01 (INS):** This is the instruction byte, which allows you to tell the card which instruction you're calling. When writing your own applet, you decide and define what values do what.

**First 00 (P1):** Parameter 1, any metadata or configuration you'd need related to the instruction. For just a simple echo applet this isn't important.

**Second 00 (P2):** P2 or parameter 2, any metadata or configuration you'd need related to the instruction. For just a simple echo applet this isn't important.

**04 (Lc):** The length of the data you're sending tied to the instruction, which would be omitted if you're not sending any data.

**01 02 03 04 (Data):** The data we're sending tied to the instruction. In context of `INS_ECHO`, this is the message we're echoing back to ourselves.

7. Hit "Send", and you should get a follow-up Response APDU. If it ends with `90 00` then that means all's good and the instruction was performed successfully. In context of `INS_ECHO`, our message should proceed the status word (the last 2 bytes), aka. `01 02 03 04`.
8. Play around with this more, and make sure the `Lc` byte is set to the right number of bytes for the message.
9. This time under Command APDU, we'll write our second instruction to case how multiple instructions can be tied to one applet. In the demo, the second instruction, `INS_TRANSPOSE`, takes the message you instead and this time sends it back in reverse. It has the `INS_BYTE` value set to `0x01`.

   Try the same message as before, `80 01 00 00 04 01 02 03 04`
10. Now hit "Send" again and the Response APDU should be different from before. Instead of `01 02 03 04` you should see `04 03 02 01`.

## Java Card Terminals

This terminal demo demonstrates how another application can interface with the Java Card by sending the APDU commands automatically rather than the manual process used in debugging the applet directly.

First, after the initial setup and connection, the terminal application sends an *INS_ECHO* APDU with "Hello, world!". It receives the response, and prints it to console.

Second, the terminal application then sends an *INS_TRANSPOSE* APDU also with "Hello, world!". This time, the message is reversed by the card itself and the response is sent back to the terminal application for it to print it out.

### Generic Terminal Applet Protocol

1. Loop through every card reader, checking for a reader that's in the CARD_INSERTION state.

2. Connect to the card (optionally specifying T=0 or T=1 mode).
3. Reference a specific logical or secure channel (in our case simply the "basic channel").
4. Select the applet by its AID.
5. ***The general workload is here, aka. any instruction(s) and protocol(s) that you define.***
6. Close the channel.
7. Wait for list of card readers to change state after loop (otherwise the application will wait as much CPU time as possible).

### Open EchoTerminal Demo

1. Open IntelliJ IDEA and skip through the various initial dialog windows (leaving the default options).
2. Click `Open` and select the EchoTerminal directory.

### Running the EchoTerminal Demo

Debug by pressing `Shift+F9` or Run by pressing `Shift+F10` (or use the icons toward the top-right or the `Run` menu in the menu bar).