

## Exercício 1

Foi descarregado o JavaCard SDK 2.2.2.

## Exercício 2

O exemplo executado, no simulador **jCardSim**, utilizou a ferramenta **APDUScriptTool** para simular a execução de comandos **APDU** num *applet Java Card*.

Criou-se o ficheiro `jcardsim.cfg` com `{index} = 0`, segundo o exemplo. Este ficheiro de configuração foi utilizado para definir o **AID** e a classe do *applet* a ser utilizado no simulador.

```
com.licel.jcardsim.card.applet.0.AID=010203040506070809
com.licel.jcardsim.card.applet.0.Class=com.licel.jcardsim.samples.HelloWorldApplet
```

Adicionalmente, criou-se o *script* `helloworld.apdu`:

```
// CREATE APPLET CMD
0x80 0xb8 0x00 0x00 0x10 0x9 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x05 0x00 0x00 0x00
// SELECT APPLET CMD
0x00 0xa4 0x00 0x00 0x09 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x02;
// TEST NOP
0x00 0x02 0x00 0x00 0x00 0x02;
// test hello world from card
0x00 0x01 0x00 0x00 0x00 0x0d;
// test echo
0x00 0x01 0x01 0x00 0x0d 0x48 0x65 0x6c 0x6c 0x6f 0x20 0x77 0x6f 0x72 0x6c 0x64 0x20 0x21 0x00
```

Este *script* foi então executado e enviou seqüências de comandos ao simulador, que respondeu com os resultados esperados, incluindo a mensagem “*Hello World*!”. Obteve-se o seguinte resultado:

```
mary-linux@mary-laptop:~/MSIlinux/HC/hardware-confiavel/labs/week5$ java -cp ~/MSIlinux/HC/hardware-confiavel/labs/week5
/lib/jcardsim-2.2.2.jar com.licel.jcardsim.utils.APDUScriptTool jcardsim.cfg helloworld.apdu
CLA: 80, INS: b8, P1: 00, P2: 00, Lc: 10, 09, 01, 02, 03, 04, 05, 06, 07, 08, 09, 05, 00, 00, 02, 0f, 0f, Le: 09, 01, 02
, 03, 04, 05, 06, 07, 08, 09, SW1: 90, SW2: 00
CLA: 00, INS: a4, P1: 00, P2: 00, Lc: 09, 01, 02, 03, 04, 05, 06, 07, 08, 09, Le: 00, SW1: 90, SW2: 00
CLA: 00, INS: 02, P1: 00, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 00, INS: 01, P1: 00, P2: 00, Lc: 00, Le: 0d, 48, 65, 6c, 6c, 6f, 20, 77, 6f, 72, 6c, 64, 20, 21, SW1: 90, SW2: 00
CLA: 00, INS: 01, P1: 01, P2: 00, Lc: 0d, 48, 65, 6c, 6c, 6f, 20, 77, 6f, 72, 6c, 64, 20, 21, Le: 0d, 48, 65, 6c, 6c, 6f
, 20, 77, 6f, 72, 6c, 64, 20, 21, SW1: 90, SW2: 00
```

Figure 1: exemplo\_cli

## Exercício 3

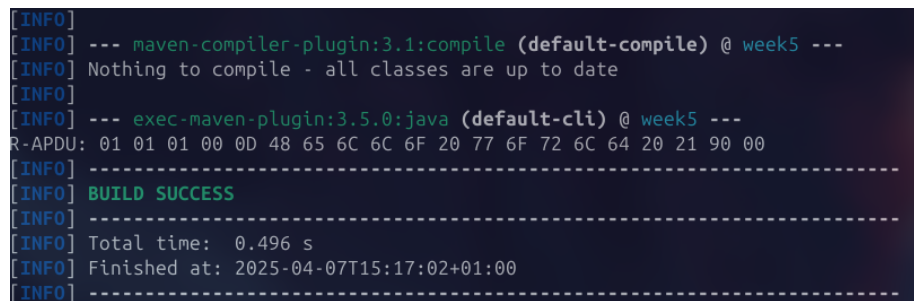
De modo a simular um **Java Card** e resolver os exercícios propostos, criou-se um projeto **Maven** com a seguinte dependência:

```
<!-- https://mvnrepository.com/artifact/com.licel/jcardsim -->
<dependency>
  <groupId>com.licel</groupId>
  <artifactId>jcardsim</artifactId>
  <version>2.2.2</version>
</dependency>
```

Para além disso, para configurar o simulador, seguiu-se a documentação do repositório oficial.

### 3

Ao compilar e executar o código do *applet* **Echo**, através do Maven, com o comando `run sim=App`, obteve-se o seguinte:



```
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ week5 ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- exec-maven-plugin:3.5.0:java (default-cli) @ week5 ---
R-APDU: 01 01 01 00 0D 48 65 6C 6C 6F 20 77 6F 72 6C 64 20 21 90 00
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.496 s
[INFO] Finished at: 2025-04-07T15:17:02+01:00
[INFO] -----
```

Figure 2: echo1

A resposta (**R-ADPU**) obtida foi a esperada, já que é possível observar a mensagem “*Hello World !*” em hexadecimal.

---

O próximo passo foi, então, modificar o ficheiro `Echo.java`, com o objetivo do *applet* manter o **número de APDU processadas** e devolver na R-APDU o **complemento binário dos dados que recebe**.

Para isso, foi adicionado um `apduCounter` que incrementa a cada APDU processada e uma linha que executa a operação **XOR** de cada *byte* com **=0xFF**:

```
for (short i = 0; i < bytesRead; i++) {
    echoBytes[echoOffset + i] = (byte) (buffer[ISO7816.OFFSET_CDATA + i] ^ (byte) 0xFF);
}
echoOffset += bytesRead;
```

Obteve-se o seguinte resultado:

A R-ADPU confirma-se pelo seguinte:

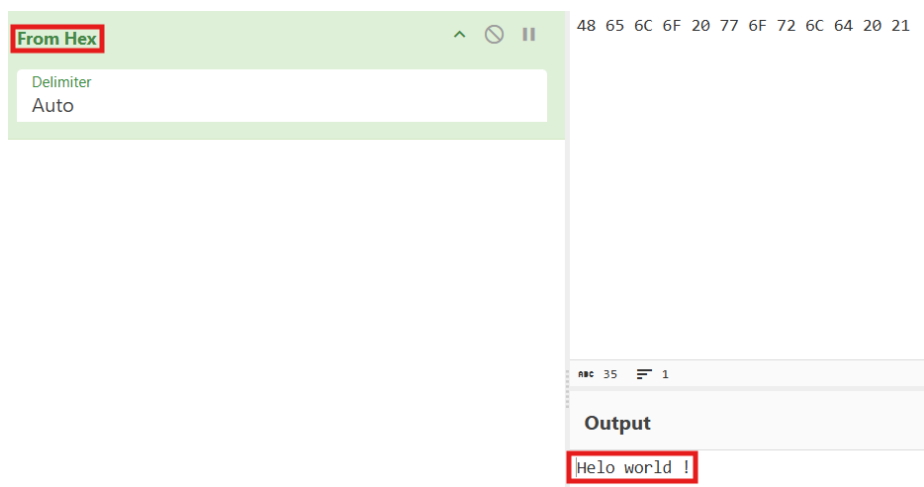


Figure 3: hex1

```
[INFO]
[INFO] --- exec-maven-plugin:3.5.0:java (default-cli) @ week5 ---
APDUs processadas: 1
APDUs processadas: 2
R-APDU: 01 01 01 00 0D B7 9A 93 93 90 DF 88 90 8D 93 9B DF DE 90 00
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.732 s
[INFO] Finished at: 2025-04-07T15:04:19+01:00
[INFO] -----
```

Figure 4: echo2

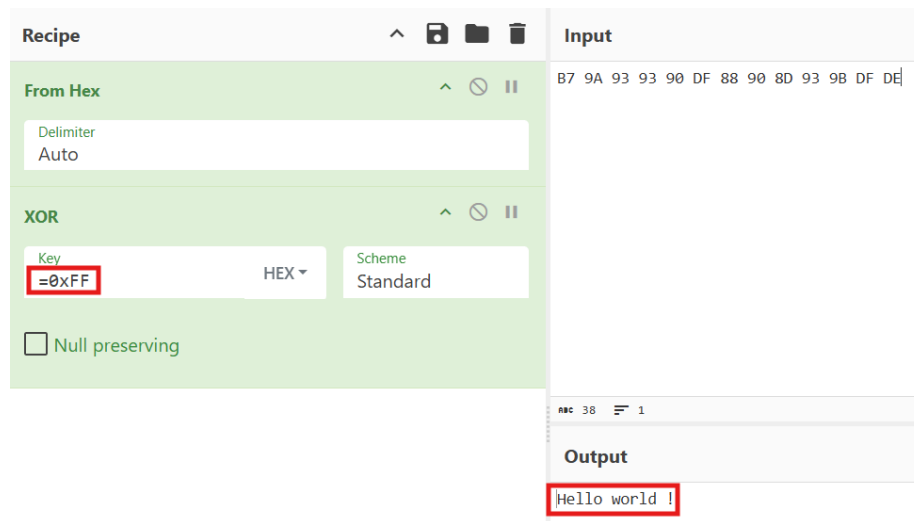


Figure 5: hex2

### 3.1

Inicialmente, foi compilado e executado o *applet* **Wallet**, através do Maven. O output obtido foi o seguinte:

No ficheiro `App.java` é criado um array `installData` com os dados da instalação do applet:

```
byte[] installData = new byte[] {
    (byte) aid.length,      // AID length (9)
    // AID (9 bytes)
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
    // PIN
    0x02,
    // PIN bytes
    0x12, 0x34
};
```

Cada linha de código abaixo simula uma ação do utilizador.

```
send(simulator, new byte[] { 0x50, 0x20, 0x00, 0x00, 0x02, 0x12, 0x34 }); // Verifica PIN c
send(simulator, new byte[] { 0x50, 0x30, 0x00, 0x00, 0x01, 0x20 }); // Credito
send(simulator, new byte[] { 0x50, 0x40, 0x00, 0x00, 0x01, 0x0F }); // Debito
send(simulator, new byte[] { 0x50, 0x50, 0x00, 0x00, 0x00 }); // Saldo
```

Depois de analisar o *applet* foi verificado o que acontece quando se manda o PIN incorreto.

```
send(simulator, new byte[] { 0x50, 0x20, 0x00, 0x00, 0x02, 0x12, 0x35 }); // Verifica PIN in
```

```

=> 50 20 00 00 02 12 34
<= 90 00

=> 50 30 00 00 01 20
<= 90 00

=> 50 40 00 00 01 0F
<= 90 00

=> 50 50 00 00 00
<= 00 11 90 00

```

Figure 6: wallet1

```

send(simulator, new byte[] { 0x50, 0x30, 0x00, 0x00, 0x01, 0x20 }); // Credito
send(simulator, new byte[] { 0x50, 0x40, 0x00, 0x00, 0x01, 0x0F }); // Debito
send(simulator, new byte[] { 0x50, 0x50, 0x00, 0x00, 0x00 }); // Saldo

```

A resposta foi a seguinte:

Assim, conseguimos ver que o resultado foi 63 00 e 63 01 que correspondem a verificação errada e verificação necessária, respetivamente. Abaixo, podemos ver a confirmação do resultado obtido, que se encontra no *applet* `Wallet.java`.

```

final static short SW_VERIFICATION_FAILED = 0x6300;
final static short SW_PIN_VERIFICATION_REQUIRED = 0x6301;

```

De seguida, verificou-se o que acontece quando se manda várias vezes o PIN incorreto e de seguida o PIN correto.

No *applet* `Wallet.java` observou-se que o número limite de tentativas são 3 e que, após 3 tentativas erradas, o PIN bloqueia.

```

final static byte PIN_TRY_LIMIT = (byte)0x03;

send(simulator, new byte[] { 0x50, 0x20, 0x00, 0x00, 0x02, 0x12, 0x35 }); // Verifica PIN i
send(simulator, new byte[] { 0x50, 0x20, 0x00, 0x00, 0x02, 0x12, 0x35 }); // Verifica PIN i
send(simulator, new byte[] { 0x50, 0x20, 0x00, 0x00, 0x02, 0x12, 0x35 }); // Verifica PIN i
send(simulator, new byte[] { 0x50, 0x20, 0x00, 0x00, 0x02, 0x12, 0x34 }); // Verifica PIN c
send(simulator, new byte[] { 0x50, 0x30, 0x00, 0x00, 0x01, 0x20 }); // Credito
send(simulator, new byte[] { 0x50, 0x40, 0x00, 0x00, 0x01, 0x0F }); // Debito
send(simulator, new byte[] { 0x50, 0x50, 0x00, 0x00, 0x00 }); // Saldo

```

Assim, com base nos resultados obtidos, podemos concluir que, após três tenta-

```

=> 50 20 00 00 02 12 35
<= 63 00

=> 50 30 00 00 01 20
<= 63 01

=> 50 40 00 00 01 0F
<= 63 01

=> 50 50 00 00 00
<= 00 00 90 00

```

Figure 7: walletpininvalid

tivas falhadas do PIN - onde todas devolvem 63 00, mesmo a mandar o PIN correto resultou, novamente, em 63 00. Isto indica que o PIN foi bloqueado após o número máximo de tentativas incorretas.

---

Foi modificado o código do *applet* de forma a que a sua inicialização aceitasse um PUK e foi alterado o valor inicial de saldo para 100.

```

byte[] installData = new byte[] {
    (byte) aid.length,      // AID length
    // AID
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
    // PIN, PUK length
    0x02, 0x02,
    // balance
    0x00, 0x64, // 100
    // PIN bytes
    0x12, 0x34,
    // PUK Bytes
    0x56, 0x78
};

```

Fizeram-se mais algumas alterações no código do *applet* Wallet para que o mesmo aceitasse o PUK e desbloqueasse o PIN. As principais alterações estão abaixo representadas e o código completo encontra-se em `Wallet.java`.

```

final static byte UNBLOCK_PIN = (byte) 0x60;
final static byte MAX_PUK_SIZE = (byte) 0x08;

```

```
=> 50 20 00 00 02 12 35
<= 63 00

=> 50 20 00 00 02 12 35
<= 63 00

=> 50 20 00 00 02 12 35
<= 63 00

=> 50 20 00 00 02 12 34
<= 63 00

=> 50 30 00 00 01 20
<= 63 01

=> 50 40 00 00 01 0F
<= 63 01

=> 50 50 00 00 00
<= 00 00 90 00
```

Figure 8: walletblocked

```

final static byte PUK_TRY_LIMIT = (byte) 0x03;

OwnerPIN puk;

puk = new OwnerPIN(PUK_TRY_LIMIT, MAX_PUK_SIZE);

case UNBLOCK_PIN:
    unblockPIN(apdu);
    return;

private void unblockPIN(APDU apdu) {
    byte[] buffer = apdu.getBuffer();
    byte len = (byte)(apdu.setIncomingAndReceive());

    if (!puk.check(buffer, ISO7816.OFFSET_CDATA, len)) {
        ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
    }
    pin.resetAndUnblock();
}

```

Após as alterações do *applet Wallet* compilou-se e executou-se o código - vai-se mandar 3 vezes o PIN incorreto, depois o PIN correto para se verificar que está bloqueado, o desbloqueio com o PUK, e por fim, o PIN correto para se verificar que o desbloqueio funcionou.

```

send(simulator, new byte[] { 0x50, 0x20, 0x00, 0x00, 0x02, 0x12, 0x35 }); // Verifica PIN in
send(simulator, new byte[] { 0x50, 0x20, 0x00, 0x00, 0x02, 0x12, 0x35 }); // Verifica PIN in
send(simulator, new byte[] { 0x50, 0x20, 0x00, 0x00, 0x02, 0x12, 0x35 }); // Verifica PIN in
send(simulator, new byte[] { 0x50, 0x20, 0x00, 0x00, 0x02, 0x12, 0x34 }); // Verifica PIN co
send(simulator, new byte[] { 0x50, 0x60, 0x00, 0x00, 0x02, 0x56, 0x78 }); // Desbloqueio con
send(simulator, new byte[] { 0x50, 0x20, 0x00, 0x00, 0x02, 0x12, 0x34 }); // Verifica PIN co
send(simulator, new byte[] { 0x50, 0x30, 0x00, 0x00, 0x01, 0x20 }); // Credito
send(simulator, new byte[] { 0x50, 0x40, 0x00, 0x00, 0x01, 0x0F }); // Debito
send(simulator, new byte[] { 0x50, 0x50, 0x00, 0x00, 0x00 }); // Saldo

```

O resultado foi o seguinte:

Conseguimos observar que o resultado é 90 00, que era o resultado certo que dava inicialmente.



```
=> 50 20 00 00 02 12 35
<= 63 00

=> 50 20 00 00 02 12 35
<= 63 00

=> 50 20 00 00 02 12 35
<= 63 00

=> 50 20 00 00 02 12 34
<= 63 00

=> 50 60 00 00 02 56 78
<= 90 00

=> 50 20 00 00 02 12 34
<= 90 00

=> 50 30 00 00 01 20
<= 90 00

=> 50 40 00 00 01 0F
<= 90 00

=> 50 50 00 00 00
<= 00 75 90 00
```

Figure 9: walletpuk