BÀI 1 Cơ chế làm việc của APDU trong Applet

1.1 Truy xuất bộ đệm APDU

JCRE làm việc với máy chủ thông qua lệnh APDU. Để xử lý APDU, trước tiên, một Applet lấy tham chiếu đến bộ đệm APDU bằng cách gọi phương thức `getBuffer`. Bộ đệm APDU là một mảng byte có độ dài có thể được xác định bằng cách sử dụng `apdu_buffer.length`.

```
// truy xuat bo dem APDU
byte[] apdu_buffer = apdu.getBuffer();
```

Phương thức trên trả ra mảng byte bộ đệm APDU, nghĩa là sau khi phương thức `getBuffer` được gọi chỉ có 5 byte đầu tiên được đẩy vào mảng byte `apdu_buffer` là: 4 byte đầu tiên là tiêu đề APDU [CLA, INS, PI, P2] và byte thứ năm (P3) là trường độ dài bổ sung. Ý nghĩa của P3 được xác định bởi từng trường hợp của lệnh:

- Đối với trường hợp 1: không có dữ liệu được truyền tới hoặc nhận từ thẻ (lệnh APDU chỉ chứa header), P3 = 0.
- Đối với trường hợp 2: không có dữ liệu được truyền tới thẻ nhưng có dữ liệu phản hồi từ thẻ, P3 = Le, độ dài của dữ liệu phản hồi.
- Đối với trường hợp 3 và 4: dữ liệu được truyền tới thẻ, P3 = Lc, độ dài của dữ liệu lệnh đến.

Để lấy ra các byte trong tiêu đề ADPU, chúng ta sử dụng các hằng số được định nghĩa trong giao diện ISO 7816:

```
apdu_buffer[OFFSET_CLA],
apdu_buffer[OFFSET_INS],
apdu_buffer[OFFSET_P1],
apdu_buffer[OFFSET_P2],
apdu_buffer[OFFSET_LC],
```

1.2 Đọc dữ liệu vào bộ đệm APDU

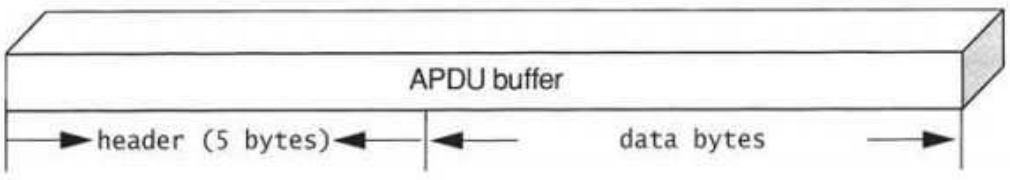
Applet có thể tìm ra số byte dữ liệu từ trường Lc (byte thứ năm trong bộ đệm APDU).

```
short data_length = (short)(apdu_buffer[ISO7816.OFFSET_LC] &
0xFF);
```

Các kiểu dữ liệu số nguyên trong ngôn ngữ lập trình Java là kiểu có dấu. Tuy nhiên, trường Lc cần được hiểu là một giá trị không dấu. Trong đoạn mã, byte Lc được kết hợp theo từng bít với hằng số 0xFF để chuyển đổi một byte có dấu thành giá trị không dấu. (Yêu cầu giải thích)

Để đọc dữ liệu vào bộ đệm APDU, Applet gọi phương thức `setIncomingAndReceive` trong lớp APDU. Phương thức `setIncomingAndReceive` hoàn thành hai nhiệm vụ:

- Đặt JCRE vào chế độ nhận dữ liệu. Đường truyền thông đến và từ thẻ thông minh là bán song công, nghĩa là dữ liệu được gửi từ máy chủ đến thẻ hoặc từ thẻ đến máy chủ nhưng không phải cả hai cùng một lúc. Bước này hướng dẫn JCRE xử lý byte thứ năm trong bộ đệm APDU như trường Lc và chuẩn bị JCRE để nhận dữ liệu đến.
- Yêu cầu JCRE nhận các byte dữ liệu lệnh đến, bắt đầu từ offset `ISO7816.OFFSET_DATA` (= 5) trong bộ đệm APDU theo tiêu đề, như trong Hình 1.1.



Hình 1.1 Bộ đệm APDU sau khi gọi phương thức `setIncomingAndReceive`

Phương thức `setIncomingAndReceive` trả về số byte mà nó đọc được. Nó trả về 0 nếu không có dữ liệu, nếu phương thức `setIncomingAndReceive` được gọi khi

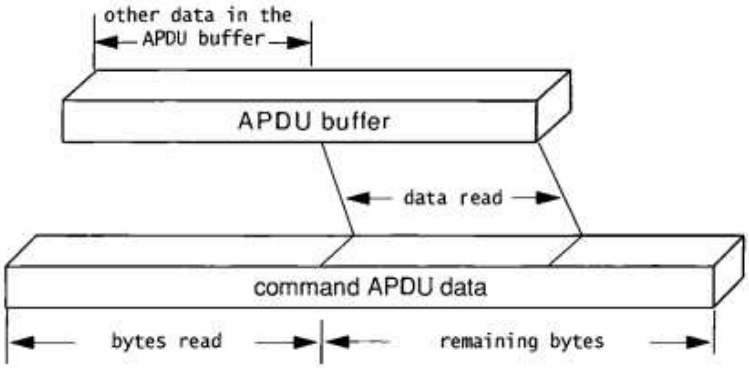
JCRE đã ở chế độ nhận từ một cuộc gọi trước đó cho cùng một phương thức, một `APDUException` xuất hiện với mã lý do `APDUException.ILLECAL_USE`

Nhận dữ liệu lệnh dài

Trong hầu hết các trường hợp, phương thức `setIncomingAndReceive` đủ để đọc số byte dữ liệu vào bộ đệm APDU như được chỉ định trong trường Lc. Tuy nhiên, đối với một lệnh APDU có nhiều dữ liệu hơn mức có thể phù hợp với bộ đệm APDU, ngoài phương thức `setIncomingAndReceive` cần sử dụng thêm một hoặc nhiều lệnh gọi đến phương thức `receiveBytes`:

```
public short receiveBytes(short bOff) throws APDUException
```

Đối với dữ liệu lệnh APDU dài, Applet có thể xử lý dữ liệu từng phần và sau đó gọi phương thức `receiveBytes` để đọc dữ liệu bổ sung vào bộ đệm APDU. Với phương thức `receiveBytes`, phần bù vào bộ đệm APDU nơi dữ liệu được nhận có thể được chỉ định. Điều này cho phép Applet kiểm soát cách sử dụng bộ đệm APDU trong quá trình xử lý dữ liệu đến. Ví dụ, như trong Hình 1.2, Applet có thể đã xử lý dữ liệu từ cuộc gọi trước đó đến phương thức `setIncomingAndReceive` hoặc phương thức `receiveBytes`, ngoại trừ một vài byte.



Hình 1.2 Gọi phương thức `receiveBytes`

Applet có thể di chuyển các byte này vào đầu bộ đệm và sau đó nhận nhóm tiếp theo để nó được gắn vào các byte vẫn còn trong bộ đệm. Tính năng này rất quan trọng trong trường hợp dữ liệu được đọc trên các yêu cầu phương thức cần được xử lý toàn bộ.

Giống như phương thức `setIncomingAndReceive`, phương thức `receiveBytes` được đảm bảo trả về đồng bộ, lấy ra càng nhiều byte càng tốt. Tuy nhiên, tùy thuộc vào số lượng byte mà máy chủ sẽ gửi dưới dạng một lô và tùy thuộc vào việc triển khai JCRE, cả hai phương thức có thể đọc được ít byte hơn không gian có sẵn trong bộ đệm APDU.

Như một quy tắc chung, cả phương thức `setIncomingAndReceive` và phương thức `receiveBytes` đều được tối ưu hóa. Nếu toàn bộ dữ liệu lệnh phù hợp trong bộ đệm APDU bắt đầu ở phần bù `ISO7816.OFFSET_CDATA` (= 5), một lệnh gọi của phương thức `setIncomingAndReceive` sẽ đủ để nhận tất cả dữ liệu. Viêc gọi đến phương thức `receiveBytes` là không cần thiết. Hầu hết các lệnh APDU thuộc loại này.

Theo tiêu chuẩn của lớp javacard.framework.APDU thì độ dài của bộ đệm APDU tối thiểu phải từ133 bytes (5 bytes tiêu đề và 128 bytes dữ liệu). Khi đó, chúng ta cần sử dụng phương thức `receiveBytes`. Nếu các byte còn lại vừa với không gian có sẵn ở phần bù được chỉ định trong bộ đệm APDU, phương thức `receiveBytes` được đảm bảo trả về với tất cả dữ liệu còn lại. Mặt khác, phương thức đọc nhiều byte sẽ phù hợp với bộ đệm và có thể ít hơn. Các Applet cần gọi `receiveBytes` nhiều lần, xử lý hoặc nhập các byte trong bộ đệm dữ liệu APDU với mỗi cuộc gọi, cho đến khi tất cả dữ liệu có sẵn được đọc. Ví dụ sau bao hàm phương thức `receiveBytes` bên trong vòng lặp `while`:

```
public void process(APDU apdu)
     {
          if (selectingApplet())
          {
               return;
          }

          byte[] buf = apdu.getBuffer();
          byte[] buf_temp = new byte[256];
          //nhan du lieu dai
```

```
            short dataLen = (short)(buf[ISO7816.OFFSET_LC]&0xff);
            short byteRead = (short)(apdu.setIncomingAndReceive());

            short pointer = 0;
            while ( dataLen > 0){
                  // copy du lieu tu buf vao bien tam buf_temp
                  Util.arrayCopy(buf, ISO7816.OFFSET_CDATA,
buf_temp, pointer, byteRead);
                  pointer += byteRead;
                  dataLen -= byteRead;
                  byteRead = apdu.receiveBytes (
ISO7816.OFFSET_CDATA ); // doc them du lieu
            }
            // thuc hien cac nhiem vu khac va tra loi may chu
// ...
      }
```

1.3 Trả về dữ liệu phản hồi APDU

Sau khi hoàn thành các chỉ thị được chỉ định trong lệnh APDU, Applet có thể trả lại dữ liệu cho máy chủ. Vì đường dẫn truyền thông cơ bản là bán song công, để gửi dữ liệu, trước tiên, Applet phải gọi phương thức `setOutgoing` để chỉ ra rằng bây giờ nó muốn gửi dữ liệu phản hồi.

```
public short setOutgoing() throws APDUException
```

Phương thức `setOutgoing` đặt JCRE sang chế độ gửi dữ liệu bằng cách đặt lại hướng truyền dữ liệu ra bên ngoài. Không giống như phương thức `setIncomingAndReceive` dùng để đọc dữ liệu, phương thức `setOutgoing` không gửi bất kỳ byte nào; nó chỉ đặt chế độ truyền dữ liệu. Khi phương thức `setOutgoing` được gọi, mọi dữ liệu đến còn lại sẽ bị loại bỏ và Applet không thể tiếp tục nhận dữ liệu.

Phương thức `setOutgoing` trả về số byte dữ liệu phản hồi (Le) mà máy chủ dự kiến cho lệnh APDU mà Applet đang phản hồi. Trong trường hợp 4 của giao thức T = 0, vì trường Le thực tế không thể được xác định, giá trị tối đa được phép cho trường Le, giả thiết là 255 byte. Đối với các trường hợp khác, trường Le thực tế được trả về trong lệnh APDU. Tuy vậy, các Applet không cần biết giao thức vận chuyển nào đang được sử dụng.

Mặc dù ứng dụng máy chủ yêu cầu trả về các byte dữ liệu Le, Applet có thể gửi nhiều hơn hoặc ít hơn số đó. Sau khi gọi phương thức `setOutgoing`, ứng dụng phải gọi phương thức `setOutgoingLength` để cho máy chủ biết có bao nhiêu byte dữ liệu phản hồi (không bao gồm SW) mà nó sẽ thực sự gửi:

```
public void setOutgoingLength(short length) throws APDUException
```

Máy chủ giả định rằng độ dài mặc định của dữ liệu phản hồi là 0, vì vậy phương thức này không cần phải được gọi nếu Applet sẽ không gửi bất kỳ dữ liệu nào. Applet không thể gửi nhiều hơn 256 byte đến máy chủ hoặc gọi phương thức `setOutgoingLength` sẽ cho ra kết quả một `APDUException` với mã lý do `APDUException.BAD_LENGTH`.

Tiếp theo, để thực sự gửi dữ liệu phản hồi, Applet gọi phương thức `sendBytes`:

```
public void sendBytes(short b0ff, short len) throws APDUException
```

Phương thức `sendBytes` gửi `len` byte dữ liệu từ bộ đệm APDU tại offset đã quy định `b0ff`. Do đó, Applet phải tạo hoặc sao chép phản hồi vào bộ đệm APDU trước khi gọi phương thức này.

Các phương thức `setOutgoing`, `setOutgoingLength` và `sendBytes` phải được gọi theo đúng thứ tự; nếu không, JCRE sẽ đưa ra một `APDUException`. Đoạn mã sau đây thể hiện công dụng của chúng trong một Applet:

```
public void process(APDU apdu)
     {
          if (selectingApplet())
          {
               return;
          }

          byte[] buf = apdu.getBuffer();
          short byteRead = (short)(apdu.setIncomingAndReceive());

          switch (buf[ISO7816.OFFSET_INS])
          {
               case (byte)0x00:

                    //gui du lieu phan hoi
                    // Buoc 1: dat JCRE sang che do gui du lieu
va nhan do dai du kien cua phan hoi (Le)
                    short le = apdu.setOutgoing();
                    if (le < (short)2) ISOException.throwIt(
ISO7816.SW_WRONG_LENGTH );

                    //Buoc 2: thong bao cho may chu biet so byte
se gui
                    apdu.setOutgoingLength( (short) (5));


                    //Buoc 3: gui du lieu
                    apdu.sendBytes((short)0, (short) (5));
                    break;
               default:

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

```
        }
    }
```

Trong ví dụ trên, lệnh APDU gửi xuống sẽ được đẩy vào bộ đệm `buf`, sau đó sẽ gửi 5 bytes tiêu đề của APDU lên cho máy chủ. Kết quả thực thi Applet như sau:

```
>> /send 00000102050102030405
>> 00 00 01 02 05 01 02 03 04 05
<< 00 00 01 02 05 90 00
```

Nếu Applet cần gửi thêm dữ liệu, nó có thể cập nhật bộ đệm APDU với dữ liệu mới và gọi phương thức `sendBytes` liên tục cho đến khi tất cả các byte được gửi.

Để giảm tài nguyên, lớp APDU cung cấp phương thức thuận tiện `setOutgoingAndSend` để gửi dữ liệu đi:

```
public void setOutgoingAndSend (short b0ff, short len) throws
APDUException
```

Phương thức `setOutgoingAndSend` kết hợp các phương thức `setOutgoing`, `setOutgoingLength` và `sendBytes` thành một truy vấn và thực hiện các tác vụ sau:

- Đặt chế độ truyền để gửi.
- Đặt độ dài dữ liệu phản hồi thành `len`.
- Gửi các byte dữ liệu phản hồi từ bộ đệm APDU tại offset `b0ff`.

Phương thức `setOutgoingAndSend` tương tự như phương thức `setIncomingAndReceive`. Cả hai phương pháp đều đặt chế độ truyền và gửi hoặc

nhận dữ liệu trong một truy vấn. Tuy nhiên, có một nhược điểm khi sử dụng phương thức setOutgoingAndSend: dữ liệu phải phù hợp hoàn toàn trong bộ đệm APDU. Nói cách khác, không giống như phương thức setIncomingAndReceive, không có thêm dữ liệu nào có thể được gửi; bộ đệm APDU cũng không thể bị thay đổi sau khi Applet gọi phương thức setOutgoingAndSend. Bởi vì hầu hết các APDU phản hồi của một Applet đều bao gồm một trường dữ liệu chứa không quá một vài byte và phù hợp với bộ đệm APDU, nên phương thức setOutgoingAndSend cung cấp cách hiệu quả nhất để đưa ra một phản hồi ngắn, đây là cách yêu cầu ít giao thức nhất.

Chúng ta viết lại ví dụ trên, bằng cách sử dụng phương thức setOutgoingAndSend:

```
public void process(APDU apdu)
     {
          if (selectingApplet())
          {
               return;
          }

          byte[] buf = apdu.getBuffer();
          switch (buf[ISO7816.OFFSET_INS])
          {
               case (byte)0x00:

                    //gui du lieu phan hoi
                    apdu.setOutgoingAndSend((short)0, (short)
(5));
                    break;
               default:

     ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
          }
```

```
    }
```

Kết quả thực thi Applet này cũng tương tự như ở ví dụ trên:

```
>> /send 00000102050102030405
>> 00 00 01 02 05 01 02 03 04 05
<< 00 00 01 02 05 90 00
```

1.4 Gửi dữ liệu từ các điểm khác nhau và gửi phản hồi dài

Để gửi một phản hồi dài, Applet có thể liên tục gọi phương thức `sendBytes` hoặc phương thức `sendBytesLong`. Trong thực tế, cả hai phương thức có thể được gọi lần lượt để gửi dữ liệu từ các vị trí khác nhau. Hai phương thức trên chỉ có thể được gọi nếu phương thức `setOutgoing` và `setOutgoingLength` được gọi trước tiên.

Chúng ta xét ví dụ: giả sử chúng ta có một Applet xử lý các thông tin sinh viên, và chúng ta cần gửi các thông tin liên quan đến sinh viên (tên, ngày sinh, …) từ các vị trí khác nhau

```
public void process(APDU apdu)
    {
        if (selectingApplet())
        {
            return;
        }
        byte[] buf = apdu.getBuffer();
        apdu.setIncomingAndReceive();
        switch (buf[ISO7816.OFFSET_INS])
        {
        case (byte)0x00:
            byte[] name = {'N', '.', 'V', '.', 'A', 'N', 'H'};
            byte[] birthday = {10, 10, 90};
```

```
                short nameLen = (short)name.length;
                short birthdayLen = (short)birthday.length;
                short totalBytes = (short)(nameLen + birthdayLen);

                //Buoc 1: dat huong truyen du lieu
                short le = apdu.setOutgoing();
                // Buoc 2: thong bao cho may chu ve so byte thuc
te trong phan hoi
                apdu.setOutgoingLength(totalBytes);
                //Buoc 3: gui du lieu
                Util.arrayCopy(name, (short)0, buf, (short)0,
nameLen);
                apdu.sendBytes((short)0, nameLen);

                Util.arrayCopy(birthday, (short)0, buf, (short)0,
birthdayLen);
                apdu.sendBytes((short)0, birthdayLen);
                break;
          default:
     ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
          }
     }
```

Trong ví dụ trên, để gửi một dữ liệu dài, chúng ta sử dụng các phương thức `sendBytes`. Để có thể gọi phương thức này liên tục, trước mỗi lần gọi chúng ta phải cập nhật lại dữ liệu của bộ đệm APDU.

Để giảm chi phí di chuyển dữ liệu, Applet có thể gọi phương thức `sendBytesLong`:

```
public void process(APDU apdu)
     {
          if (selectingApplet())
          {
```

```
            return;
        }

        byte[] buf = apdu.getBuffer();
        apdu.setIncomingAndReceive();
        switch (buf[ISO7816.OFFSET_INS])
        {
        case (byte)0x00:
            byte[] name = {'N', '.', 'V', '.', 'A', 'N', 'H'};
            byte[] birthday = {10, 10, 90};
            short nameLen = (short)name.length;
            short birthdayLen = (short)birthday.length;
            short totalBytes = (short)(nameLen + birthdayLen);

            //Buoc 1: dat huong truyen du lieu
            short le = apdu.setOutgoing();
            // Buoc 2: thong bao cho may chu ve so byte thuc
te trong phan hoi
            apdu.setOutgoingLength(totalBytes);
            //Buoc 3: gui du lieu
            //Cach 1: su dung phuong thuc sendBytes
            // Util.arrayCopy(name, (short)0, buf, (short)0,
nameLen);
            // apdu.sendBytes((short)0, nameLen);

            // Util.arrayCopy(birthday, (short)0, buf,
(short)0, birthdayLen);
            // apdu.sendBytes((short)0, birthdayLen);

            //Cach 2: su dung phuong thuc sendBytesLong
            apdu.sendBytesLong(name, (short)0, nameLen);
```

```
                apdu.sendBytesLong(birthday, (short)0,
birthdayLen);

                break;
            default:

      ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
            }
      }
```

Kết quả thực thi của Applet viết theo 2 cách trên đều như sau:

```
>> /send 00000102
>> 00 00 01 02
<< 4E 2E 56 2E 41 4E 48 0A 0A 5A 90 00
```

Ngoài ra, như ta đã biết, phương thức `sendBytes` hoặc phương thức `sendBytesLong` gửi dữ liệu và trả về đồng bộ, do đó Applet có thể cập nhật ngay bộ đệm APDU. Tuy nhiên, Applet không nên thay đổi bộ đệm APDU sau lần gọi cuối cùng cho một phương thức như vậy. Bởi vì nếu Applet cần thực hiện các tác vụ khác trước khi phương thức `process` trả về, nó sẽ giữ nguyên bộ đệm APDU hoặc phần dữ liệu cuối cùng có thể bị hỏng khi được gửi đi. Do vậy, để gửi dữ liệu dài từ các địa điểm khác nhau, chúng ta nên sử dụng phương thức `sendBytesLong.`

1.5 Ví dụ

Bây giờ chúng ta sẽ thực hành viết một Applet máy tính bỏ túi với các chức năng cơ bản là cộng, trừ, nhân, chia sử dụng các kiến thức đã học trong bài.

Để viết Applet này, chúng ta cần thực hiện theo các bước sau:

- Bước 1. Xác định các chức năng chính của Applet: cộng, trừ, nhân, chia với tham số đầu vào được truyền thông qua P1 và P2 của trường tiêu đề
- Bước 2. Xác định AID của gói (package) và Applet: một AID gồm 2 phần RID (5 bytes) và PIX (0-11 bytes), để đơn giản chúng ta chọn RID là 0x11 0x22 0x33 0x44 0x55, PIX của gói là 0x00, PIX của Applet là 0x00 0x00 (các Applet tiếp theo của gói này sẽ có PIX tương ứng là 0x00 0x01, 0x00 0x02, …)
- Bước 3. Xác định CLA của Applet, ví dụ: 0xA0; các lệnh INS:
    - INS_CONG: 0x00
    - INS_TRU: 0x01
    - INS_NHAN: 0x02
    - INS_CHIA: 0x03
- Bước 4. Viết Applet

```
package bai1.MTBT;

import javacard.framework.*;

public class Calculator extends Applet
{
     // khai bao ma CLA cua applet
     final static byte Calculator_CLA = (byte)0xA0;

     // khai bao ma INS trong tieu de cua apdu
     final static byte INS_CONG    = (byte)0x00;
     final static byte INS_TRU     = (byte)0x01;
     final static byte INS_NHAN    = (byte)0x02;
     final static byte INS_CHIA    = (byte)0x03;
```

```
    public static void install(byte[] bArray, short bOffset,
byte bLength)
    {
        new Calculator().register(bArray, (short) (bOffset +
1), bArray[bOffset]);
    }

    public void process(APDU apdu)
    {
        if (selectingApplet())
        {
            return;
        }

        byte[] buf = apdu.getBuffer();

        // dat apdu o che do nhan du lieu
        apdu.setIncomingAndReceive();


        // kiem tra xem CLA nhap vao co hop le hay khong
        if (buf[ISO7816.OFFSET_CLA] != Calculator_CLA)

    ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
        short res = 0;
        short tmp1 = buf[ISO7816.OFFSET_P1];
        short tmp2 = buf[ISO7816.OFFSET_P2];
        switch (buf[ISO7816.OFFSET_INS])
        {
        case INS_CONG:
            res  = (short)(tmp1 + tmp2);
```

```
            break;
        case INS_TRU:
            res  = (short)(tmp1 - tmp2);


            break;
        case INS_NHAN:
            res  = (short)(tmp1 * tmp2);

            break;
        case INS_CHIA:
            res  = (short)(tmp1 / tmp2);


            break;
        default:

    ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
        }

        Util.setShort(buf, (short)0, res);

        // gui du lieu len may chu

        // dat apdu sang che do gui du lieu
        short le = apdu.setOutgoing();
        if (le < (short)2) ISOException.throwIt(
ISO7816.SW_WRONG_LENGTH );

        // gui cho may chu so byte se gui
        apdu.setOutgoingLength( (short)2 );
```

```
            // gui du lieu cho may chu
            apdu.sendBytes((short)0, (short)2);


            /*Chu y: doi voi mot phan hoi ngan nhu trong truong hop
nay,
            chung ta co the ket hop 3 phuong thuc
                  apdu.setOutgoing(),
                  apdu.setOutgoingLength()
                  apdu.sendBytes()
            bang 1 phuong thuc duy nhat
                  apdu.setOutgoingAndSend() */

            // apdu.setOutgoingAndSend((short)0, (short)2);

      }
}
```

- Bước 5. Chạy thử Applet: sau khi viết xong chúng ta tiến hành chạy thử để kiểm tra các chức năng của Applet vừa viết

```
>> /select 11223344550000
>> 00 A4 04 00 07 11 22 33 44 55 00 00 00
<< 90 00

>> /send a0000203
>> A0 00 02 03
<< 00 05 90 00

>> /send a0010502
>> A0 01 05 02
<< 00 03 90 00

>> /send a0020203
>> A0 02 02 03
<< 00 06 90 00
```

```
>> /send a0030903
>> A0 03 09 03
<< 00 03 90 00
```