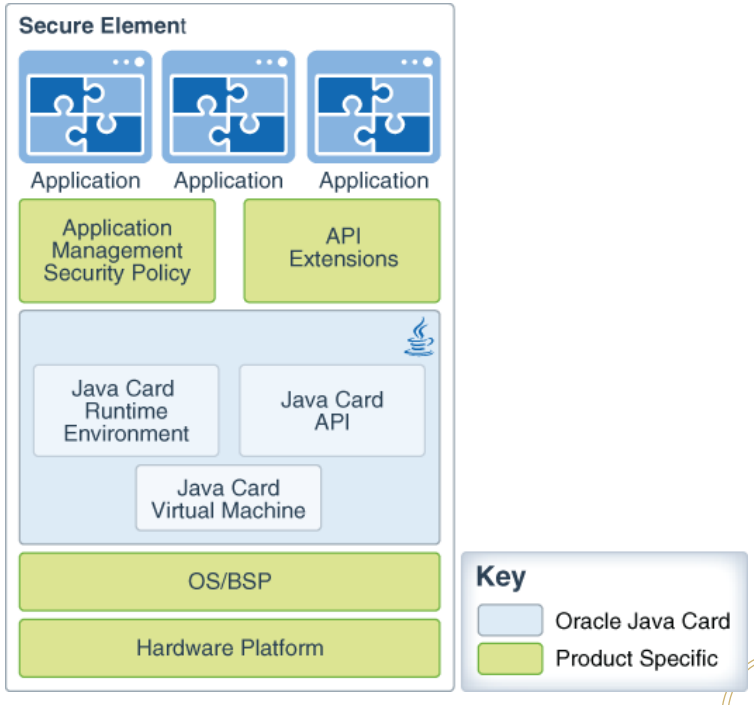
Program „Hello World”
dla Java Card 3 Connected

Edyta Bosacka 117185

Budowa Java Card OS

Na technologię Java Card składają się następujące komponenty:

- Maszyna wirtualna (Java Card Virtual Machine)
- Środowisko uruchomieniowe (Java Card Runtime Environment)
- Interfejs API (Java Card Application Programming Interface)



Architektura platformy Java Card

Budowa Java Card OS

Maszyna wirtualna (Java Card Virtual Machine)

Wirtualną maszynę Java definiuje się jako maszynę, która ładuje pliki klasy Java i wykonuje je z określonym zestawem semantyki. Pełna implementacja maszyny Java jest jednak zbyt duża, aby zmieściła się na urządzeniach, o ograniczonych zasobach pamięci, jakimi są karty elektroniczne.

Stąd maszyna wirtualna Java Card stanowi jedynie zawężony podzbiór maszyny wirtualnej Java.

Budowa Java Card OS

Maszyna wirtualna (Java Card Virtual Machine)

Maszyna wirtualna Java Card nie posiada zatem wielu elementów, które są zaimplementowane w normalnej maszynie wirtualnej Java. Należą do nich:

- dynamiczne ładowanie klas
- Security Manager
- finalizacja
- Wielowątkowość
- klonowanie obiektów
- typy enum
- ulepszona pętla for

Budowa Java Card OS

Maszyna wirtualna (Java Card Virtual Machine)

Java Card VM nie obsługuje również typów zmiennych:

- char
- double
- float
- long
- tablice wielowymiarowe

Budowa Java Card OS

Maszyna wirtualna (Java Card Virtual Machine)

Do elementów zaimplementowanych w maszynie wirtualnej Java Card należą z kolei:

- pakiety
- dynamiczne tworzenie obiektów
- metody wirtualne
- interfejsy
- wyjątki
- struktury generyczne
- import statyczny

Budowa Java Card OS

Maszyna wirtualna (Java Card Virtual Machine)

Java Card VM obsługuje również następujące typy zmiennych:

- boolean
- byte
- short
- int (opcjonalnie)
- obiekty

Budowa Java Card OS

Środowisko uruchomieniowe (Java Runtime Enivronment)

Java Card Runtime Environment stanowi zestaw elementów niezbędnych do uruchomienia apletów napisanych w Java Card. Środowisko to odpowiada za wiele aspektów pracy: nadzoruje cykl życia apletu, obsługuje kanały logiczne, zapewnia bezpieczeństwo i atomowość przeprowadzanych operacji oraz dostarcza wiele funkcji i możliwości, z których aplet może korzystać.

Budowa Java Card OS

Środowisko uruchomieniowe (Java Runtime Enivronment)

Cykl życia instancji apletu rozpoczyna się, gdy zostanie on pomyślnie zarejestrowany w Java Card RE za pomocą metody **Applet.register**. Aplety zarejestrowane za pomocą tej metody istnieją aż do momentu usunięcia ich przez Menadżera usuwania apletów.

Java Card RE inicjuje interakcje z apletem za pomocą publicznych metod apletu: `install`, `select` , `deselect` i `process`.

Budowa Java Card OS

Środowisko uruchomieniowe (Java Runtime Enivronment)

a) `Install` – głównym zadaniem metody install jest utworzenie nowej instancji apletu za pomocą jej konstruktora oraz zarejestrowanie tej instancji (metoda Applet.register).
b) `Select` – metoda ta pozwala na wybranie apletu, dzięki czemu kolejne komendy APDU będą przesyłane bezpośrednio do metody process.
c) `Deselect` – dezaktywuje aktualnie wybrany aplet. Umożliwia jednak wykonanie pewnych operacji kończących aktualną sesję.
d) `Process` – powoduje przesłanie do apletu komendy APDU i jej obsłużenie.

Budowa Java Card OS

Środowisko uruchomieniowe (Java Runtime Enivronment)

Java Card w wersji 3 posiada wsparcie dla obsługi kanałów logicznych. Pozwala ono na otwarcie maksymalnie 20 połączeń poprzez dowolny terminal wejścia/wyjścia (I/O).

Funkcjonalność kanałów logicznych pozwala m.in. na obsługę wielu sesji, równoległy wybór kilku apletów lub kilkukrotny wybór jednego apletu na różnych kanałach jednocześnie.

Budowa Java Card OS

Środowisko uruchomieniowe (Java Runtime Enivronment)

Inną dostępną funkcją w Java Card RE są obiekty tymczasowe (ang. transient object), które istnieją jedynie przez określony czas. Po zakończeniu wykonywania apletu (np. wybranie innego apletu) ich wartość bowiem jest kasowana.

Obiekty tymczasowe są idealne dla niewielkich ilości tymczasowych danych apletów, które są często modyfikowane i nie muszą być danymi trwałymi podczas sesji CAD (Card Acceptance Device).

Budowa Java Card OS

Interfejs API (Java Card Application Programming Interface)

Podczas tworzenia aplikacji dla kart najczęściej korzysta się z następujących pakietów języka Java:

- **java.lang** – zawiera podzbiór podstawowych klas języka Java przeznaczonych dla kart
- **java.io** – wykorzystywany jest jedynie podzbiór tego pakietu związany z wyjątkami
- **java.rmi** – pakiet używany do zdefiniowania zdalnego interfejsu karty czyli metod wywoływanych przez CAD (Card Acceptance Device)

Budowa Java Card OS

Interfejs API (Java Card Application Programming Interface)

- **javacard.framework** – dostarcza strukturę klas i interfejsów używanych przy tworzeniu i komunikowaniu się z apletami kartowymi
- **javacard.framework.Applet** – jest to abstrakcyjna klasa bazowa dla wszystkich apletów
- **javacard.framework.JCsystem** – zawiera szereg statycznych metod przeznaczonych do obsługi wykonywanego apletu, zarządzania zasobami, współdzielenia obiektów pomiędzy aplikacjami, usuwania obiektów oraz realizacji transakcji atomowych.

Budowa Java Card OS

Interfejs API (Java Card Application Programming Interface)

- **javacard.framework.AID** – używana jest do przechowywania AID (ang. application identifier) aplikacji
- **javacard.framework.Util** – zawiera zestaw przydatnych statycznych metod (kopiowanie i porównywanie tablic, konkatenacja i rozdzielanie bajtów)
- **javacard.framework.OwnerPIN** – klasa przeznaczona do przechowywania i zarządzania kodem PIN właściciela karty; zawiera mechanizmy chroniące kod PIN tj. licznik niepoprawnych wprowadzeń oraz mechanizm blokujący

Środowisko GlobalPlatform

GlobalPlatform definiuje elastyczną i wydajną specyfikację wspólną dla wszystkich wydawców kart elektronicznych.

Specyfikacja pozwala im wybrać technologię, które obecnie potrzebują, jednocześnie zapewniając możliwość ewentualnej migracji na inną technologię w przyszłości, bez znaczących zmian w infrastrukturze karty.

Środowisko GlobalPlatform

Do podstawowych komponentów, z jakich złożona jest karta zgodna z GlobalPlatform należą:

- środowisko uruchomieniowe złożone z systemu operacyjnego karty, maszyny wirtualnej oraz API umożliwiającego dostęp do systemu operacyjnego z poziomu aplikacji wykonywanych przez maszynę wirtualną
- API GlobalPlatform, które jest zestawem komend pozwalającym na zarządzanie kartami wieloaplikacyjnymi podczas ich eksploatacji oraz na korzystanie z kluczy kryptograficznych do zabezpieczenia przesyłanych informacji w ramach aplikacji od jednego dostawcy

Środowisko GlobalPlatform

- zarządca karty (ang. Card manager), który jest odpowiedzialny za:
  * przekazywanie komend do określonych aplikacji,
  * zarządzanie zawartościami (aplikacjami) w karcie,
  * zarządzanie bezpieczeństwem aplikacji (przechowywanie współdzielonych kodów PIN oraz kluczy kryptograficznych)
  * zarządzanie domenami bezpieczeństwa (ang. security domains)

- domeny bezpieczeństwa, które umożliwiają logiczne rozdzielenie zasobów współdzielonych przez określone aplikacje

- aplikacje kartowe o różnej funkcjonalności

Środowisko GlobalPlatform

GlobalPlatform szczegółowo określa jakie komendy i w jaki sposób powinny być zaimplementowane w karcie. Należą do nich m.in.:

- `DELETE` – usunięcie instancji aplikacji, aplikacji lub klucza z karty
- `GET DATA` – pobranie danych o określonych obiektach z karty (np. numer identyfikacyjny wydawcy, informacje o kluczach)
- `GET STATUS` – umożliwia odczytanie aktualnego stanu obiektów takich jak aplikacja czy plik wykonywalny (zwracane są np. informacje o AID aplikacji i jej przywilejach)
- `INSTALL` – przeznaczona do zarządzania zawartością karty np. poprzez tworzenie instancji apletów w karcie

Środowisko GlobalPlatform

- `LOAD` – wykorzystywana do przekazywania bloków danych do karty
- `MANAGE CHANNEL` – przeznaczona do zarządzania komunikacją z wykorzystaniem kanałów logicznych
- `PUT KEY` – pozwala na umieszczenie w karcie nowego klucza/y lub podmianę starego klucza/y na jego nową wersję
- `SELECT` – pozwala wybrać aktualną aplikację z karty poprzez jej AID
- `SET STATUS` – przeznaczona do zarządzania cyklem życia karty lub aplikacji
- `INITIALIZE UPDATE` – inicjalizacja kanału do zabezpieczonego przesyłania danych

Możliwości wykonania appletu w Java Card 2.2.2

```java
package com.sun.javacard.samples.HelloWorld;

import javacard.framework.*;


public class HelloWorld extends Applet
{
    private byte[] echoBytes;
    private static final short LENGTH_ECHO_BYTES = 256;

    /**
     * Only this class's install method should create the applet object.
     */
    protected HelloWorld()
    {
        echoBytes = new byte[LENGTH_ECHO_BYTES];
        register();
    }

    /**
     * Installs this applet.
     * @param bArray the array containing installation parameters
     * @param bOffset the starting offset in bArray
     * @param bLength the length in bytes of the parameter data in bArray
     */
    public static void install(byte[] bArray, short bOffset, byte bLength)
    {
        new HelloWorld();
    }

    /**
     * Processes an incoming APDU.
     * @see APDU
     * @param apdu the incoming APDU
     * @exception ISOException with the response bytes per ISO 7816-4
     */
    public void process(APDU apdu)
    {
        byte buffer[] = apdu.getBuffer();

        short bytesRead = apdu.setIncomingAndReceive();
        short echoOffset = (short)0;

        while ( bytesRead > 0 ) {
            Util.arrayCopyNonAtomic(buffer, ISO7816.OFFSET_CDATA, echoBytes, echoOffset, bytesRead);
            echoOffset += bytesRead;
            bytesRead = apdu.receiveBytes(ISO7816.OFFSET_CDATA);
        }

        apdu.setOutgoing();
        apdu.setOutgoingLength( (short) (echoOffset + 5) );

        // echo header
        apdu.sendBytes( (short)0, (short) 5);
        // echo data
        apdu.sendBytesLong( echoBytes, (short) 0, echoOffset );
    }
}
```

Omówienie kodu

Metoda process

1. Pobranie bufora APDU (Application Protocol Data Unit)
2. Ustawienie APDU w tryb odczytywania danych
3. W pętli while:
   a) wczytanie danych oraz offsetu i zapisanie ich w polu echoBytes
4. Zmiana trybu APDU na wysyłanie danych
5. Wysyłanie nagłówka wiadomości
6. Wysłanie przez aplet tej samej wiadomości, którą otrzymał wcześniej.

```
public void process(APDU apdu)
{
    byte buffer[] = apdu.getBuffer();

    short bytesRead = apdu.setIncomingAndReceive();
    short echoOffset = (short)0;

    while ( bytesRead > 0 ) {
        Util.arrayCopyNonAtomic(buffer, ISO7816.OFFSET_CDATA, echoBytes, echoOffset, bytesRead);
        echoOffset += bytesRead;
        bytesRead = apdu.receiveBytes(ISO7816.OFFSET_CDATA);
    }

    apdu.setOutgoing();
    apdu.setOutgoingLength( (short) (echoOffset + 5) );

    // echo header
    apdu.sendBytes( (short)0, (short) 5);
    // echo data
    apdu.sendBytesLong( echoBytes, (short) 0, echoOffset );
}
```

Możliwości wykonania appletu w Java Card 3.1

```java
package com.oracle.jcclassic.samples.helloworld;

import javacard.framework.APDU;
import javacard.framework.Applet;
import javacard.framework.ISO7816;
import javacard.framework.ISOException;
import javacard.framework.Util;

/**
 */

public class HelloWorld extends Applet {
    private byte[] echoBytes;
    private static final short LENGTH_ECHO_BYTES = 256;

    /**
     * Only this class's install method should create the applet object.
     */
    protected HelloWorld() {
        echoBytes = new byte[LENGTH_ECHO_BYTES];
        register();
    }

    /**
     * Installs this applet.
     *
     * @param bArray
     *            the array containing installation parameters
     * @param bOffset
     *            the starting offset in bArray
     * @param bLength
     *            the length in bytes of the parameter data in bArray
     */
    public static void install(byte[] bArray, short bOffset, byte bLength) {
        new HelloWorld();
    }

    /**
     * Processes an incoming APDU.
     *
     * @see APDU
     * @param apdu
     *            the incoming APDU
     * @exception ISOException
     *                with the response bytes per ISO 7816-4
     */
    @Override
    public void process(APDU apdu) {
        byte buffer[] = apdu.getBuffer();

        // check SELECT APDU command
        if ((buffer[ISO7816.OFFSET_CLA] == 0) &&
                (buffer[ISO7816.OFFSET_INS] == (byte) (0xA4))) {
            return;
        }

        short bytesRead = apdu.setIncomingAndReceive();
        short echoOffset = (short) 0;

        while (bytesRead > 0) {
            Util.arrayCopyNonAtomic(buffer, ISO7816.OFFSET_CDATA, echoBytes, echoOffset, bytesRead);
            echoOffset += bytesRead;
            bytesRead = apdu.receiveBytes(ISO7816.OFFSET_CDATA);
        }

        apdu.setOutgoing();
        apdu.setOutgoingLength((short) (echoOffset + 5));

        // echo header
        apdu.sendBytes((short) 0, (short) 5);
        // echo data
        apdu.sendBytesLong(echoBytes, (short) 0, echoOffset);
    }
}
```

Omówienie kodu

Metoda process

1. Pobranie bufora APDU (Application Protocol Data Unit)
2. Sprawdzenie, czy poleceniem w buforze jest SELECT FILE. Jeśli tak, to nastąpi wyjście z metody process.

```
// check SELECT APDU command
if ((buffer[ISO7816.OFFSET_CLA] == 0) &&
        (buffer[ISO7816.OFFSET_INS] == (byte) (0xA4))) {
    return;
}
```

W przeciwnym razie wykona się:

3. Ustawienie APDU w tryb odczytywania danych
4. W pętli while:
   a) wczytanie danych oraz offsetu i zapisanie ich w polu echoBytes
5. Zmiana trybu APDU na wysyłanie danych
6. Wysyłanie nagłówka wiadomości
7. Wysłanie przez aplet tej samej wiadomości, którą otrzymał wcześniej.

Możliwości wykonania appletu w Java Card 3.05 Classic oraz Connected

Java Card 3.05 Classic

```java
package com.oracle.jcclassic.samples.helloworld;

import javacard.framework.APDU;
import javacard.framework.Applet;
import javacard.framework.ISO7816;
import javacard.framework.ISOException;
import javacard.framework.Util;

/**
 */

public class HelloWorld extends Applet {
    private byte[] echoBytes;
    private static final short LENGTH_ECHO_BYTES = 256;

    /**
     * Only this class's install method should create the applet object.
     */
    protected HelloWorld() {
        echoBytes = new byte[LENGTH_ECHO_BYTES];
        register();
    }

    /**
     * Installs this applet.
     *
     * @param bArray
     *            the array containing installation parameters
     * @param bOffset
     *            the starting offset in bArray
     * @param bLength
     *            the length in bytes of the parameter data in bArray
     */
    public static void install(byte[] bArray, short bOffset, byte bLength) {
        new HelloWorld();
    }

    /**
     * Processes an incoming APDU.
     *
     * @see APDU
     * @param apdu
     *            the incoming APDU
     * @exception ISOException
     *                with the response bytes per ISO 7816-4
     */
    @Override
    public void process(APDU apdu) {
        byte buffer[] = apdu.getBuffer();

        // check SELECT APDU command
        if ((buffer[ISO7816.OFFSET_CLA] == 0) &&
                (buffer[ISO7816.OFFSET_INS] == (byte) (0xA4))) {
            return;
        }

        short bytesRead = apdu.setIncomingAndReceive();
        short echoOffset = (short) 0;

        while (bytesRead > 0) {
            Util.arrayCopyNonAtomic(buffer, ISO7816.OFFSET_CDATA, echoBytes, echoOffset, bytesRead);
            echoOffset += bytesRead;
            bytesRead = apdu.receiveBytes(ISO7816.OFFSET_CDATA);
        }

        apdu.setOutgoing();
        apdu.setOutgoingLength((short) (echoOffset + 5));

        // echo header
        apdu.sendBytes((short) 0, (short) 5);
        // echo data
        apdu.sendBytesLong(echoBytes, (short) 0, echoOffset);
    }
}
```

Java Card 3.05 Connected

```java
package com.sun.jcclassic.samples.helloworld;

import javacard.framework.APDU;
import javacard.framework.Applet;
import javacard.framework.ISO7816;
import javacard.framework.ISOException;
import javacard.framework.Util;

/**
 */

public class HelloWorld extends Applet {
    private byte[] echoBytes;
    private static final short LENGTH_ECHO_BYTES = 256;

    /**
     * Only this class's install method should create the applet obj
     */
    protected HelloWorld() {
        echoBytes = new byte[LENGTH_ECHO_BYTES];
        register();
    }

    /**
     * Installs this applet.
     *
     * @param bArray
     *            the array containing installation parameters
     * @param bOffset
     *            the starting offset in bArray
     * @param bLength
     *            the length in bytes of the parameter data in bArray
     */
    public static void install(byte[] bArray, short bOffset, byte bLength) {
        new HelloWorld();
    }

    /**
     * Processes an incoming APDU.
     *
     * @see APDU
     * @param apdu
     *            the incoming APDU
     * @exception ISOException
```

```java
    public void process(APDU apdu) {
        byte[] buffer = apdu.getBuffer();

        // check SELECT APDU command
        if ((buffer[ISO7816.OFFSET_CLA] == 0) &&
                (buffer[ISO7816.OFFSET_INS] == (byte) (0xA4))) {
            return;
        }

        apdu.setIncomingAndReceive();

        byte nameLength = buffer[ISO7816.OFFSET_LC];
        byte[] greeting = {(byte)'H',(byte)'e',(byte)'l',(byte)'l',(byte)'o',(byte)'!',(byte)' '};

        short offset = Util.arrayCopy(buffer, ISO7816.OFFSET_CDATA, buffer,
                (short)(ISO7816.OFFSET_CDATA+greeting.length), nameLength);
        Util.arrayCopy(greeting, (short)0, buffer, ISO7816.OFFSET_CDATA, (byte)(greeting.length));

        apdu.setOutgoing();
        apdu.setOutgoingLength((byte)(nameLength+greeting.length));
        apdu.sendBytes(ISO7816.OFFSET_CDATA, (byte)(nameLength+greeting.length));
    }
}
```

Omówienie kodu – różnice między Classic a Connected

1. W wersji Connected wiadomość „Hello!” jest zdefiniowana bezpośrednio w aplecie.

```
byte [] greeting = {(byte)'H', (byte)'e', (byte)'l', (byte)'l', (byte)'o', (byte)'!', (byte)''};
```

Natomiast w Classic odpowiedzią jest odesłanie otrzymanej wiadomości.

(najpierw do echoBytes zapisywana jest odebrana wiadomość - `Util.arrayCopyNonAtomic(buffer,ISO7816OFFSET_CDATA,echoBytes,echoOffset, bytesRead);`

następnie w sendBytes odsyłane jest echoBytes z powrotem)

Omówienie kodu – różnice między Classic a Connected

2. W wersji Connected kopiowanie wczytanych danych odbywa się za jednym wykonaniem metody **Util.arrayCopy()**.

```
short offset = Util.arrayCopy(buffer, ISO7816.OFFSET_CDATA, buffer,
                              (short)(ISO7816.OFFSET_CDATA+greeting.length), nameLength);
Util.arrayCopy(greeting, (short)0, buffer, ISO7816.OFFSET_CDATA, (byte)(greeting.length));
```

W wersji Classic natomiast wiadomość może być odczytywana w częściach w pętli while i przy użyciu metody **Util.arrayCopyNonAtomic()**.

```
while (bytesRead > 0) {
    Util.arrayCopyNonAtomic(buffer, ISO7816.OFFSET_CDATA, echoBytes, echoOffset, bytesRead);
    echoOffset += bytesRead;
    bytesRead = apdu.receiveBytes(ISO7816.OFFSET_CDATA);
}
```

Omówienie kodu – różnice między Classic a Connected

3. W wersji Connected wiadomość jest wysyłana razem z offsetem jedną metodą **apdu.sendBytes()**.

```
apdu.setOutgoing();
apdu.setOutgoingLength((byte)(nameLength+greeting.length));
apdu.sendBytes(ISO7816.OFFSET_CDATA, (byte)(nameLength+greeting.length));
```

Natomiast w wersji Classic najpierw wywoływana jest najpierw metoda **apdu.sendBytes()** w celu wysłania offsetu, a następnie metoda **sendBytesLong()**, która wysyła właściwą odpowiedź.

```
// echo header
apdu.sendBytes((short) 0, (short) 5);
// echo data
apdu.sendBytesLong(echoBytes, (short) 0, echoOffset);
```

Dziękuję za uwagę