TP Java n°1 : ATR & Système de fichiers d'une carte à puce

Thème

Communication carte–lecteur (ATR) et modélisation du système de fichiers (MF/DF/EF) en Java

Objectifs pédagogiques

À la fin de ce TP, vous serez capables de :

1. 1. Manipuler un ATR (Answer To Reset) sous forme de tableau d'octets en Java.
2. 2. Écrire une petite classe Java pour analyser un ATR simple et en extraire TS, T0, octets d'interface et octets historiques.
3. 3. Modéliser la hiérarchie MF / DF / EF d'une carte à puce en programmation orientée objet.
4. 4. Faire le lien entre la théorie du cours (chapitre 3) et une implémentation Java.

Ce TP est guidé : suivez les étapes dans l'ordre, recopiez les codes dans votre IDE, exécutez, puis répondez aux questions.

0. Pré-requis techniques

• Java installé (JDK 8 ou plus).

• Un IDE ou éditeur : IntelliJ / Eclipse / NetBeans / VS Code ou simple terminal + javac/java.

• Savoir créer un projet Java simple, créer une classe avec une méthode main.

Partie A : Manipulation et analyse d'un ATR en Java

A.1. Création du projet

5. 1. Créez un nouveau projet Java appelé par exemple : **TP_CarteAPuce_ATR**.
6. 2. Créez une classe Java nommée **AtrParser** dans un package (par exemple **tp.smartcard**).

A.2. Représenter un ATR en Java

Nous allons utiliser l'ATR simplifié suivant (fictif) :

```
3B 95 11 00 00 73 C8 40 13 00
```

Ajoutez le code suivant dans la classe **AtrParser** :

```
public class AtrParser {

    public static void main(String[] args) {
        // ATR simplifié (exemple)
        String atrHex = "3B 95 11 00 00 73 C8 40 13 00";
```

```
        // Découper la chaîne en octets
        String[] parts = atrHex.split("\\s+");
        byte[] atr = new byte[parts.length];
        for (int i = 0; i < parts.length; i++) {
            atr[i] = (byte) Integer.parseInt(parts[i], 16);
        }

        // Appeler la méthode d'analyse
        parseAndPrint(atr);
    }
```

Ajoutez ensuite la méthode suivante dans la même classe :

```
    /**
     * Analyse un ATR simplifié et affiche TS, T0,
     * les octets d'interface et les octets historiques.
     */
    public static void parseAndPrint(byte[] atr) {
        if (atr == null || atr.length < 2) {
            System.out.println("ATR trop court !");
            return;
        }

        int index = 0;
        byte TS = atr[index++];
        byte T0 = atr[index++];

        // partie basse (4 bits) = nb d'octets historiques
        int k = T0 & 0x0F;
        // partie haute (4 bits) = présence de TA1/TB1/TC1/TD1
        int y = (T0 & 0xF0) >> 4;

        Byte TA1 = null, TB1 = null, TC1 = null, TD1 = null;

        if ((y & 0x1) != 0 && index < atr.length) TA1 = atr[index++];
        if ((y & 0x2) != 0 && index < atr.length) TB1 = atr[index++];
        if ((y & 0x4) != 0 && index < atr.length) TC1 = atr[index++];
        if ((y & 0x8) != 0 && index < atr.length) TD1 = atr[index++];

        int remaining = atr.length - index;
        int histLen = Math.min(k, remaining);
        byte[] historical = new byte[histLen];
        System.arraycopy(atr, index, historical, 0, histLen);

        // --- Affichage lisible ---
        System.out.printf("TS = %02X%n", TS);
        System.out.printf("T0 = %02X%n", T0);
        if (TA1 != null) System.out.printf("TA1 = %02X%n", TA1);
        if (TB1 != null) System.out.printf("TB1 = %02X%n", TB1);
        if (TC1 != null) System.out.printf("TC1 = %02X%n", TC1);
        if (TD1 != null) System.out.printf("TD1 = %02X%n", TD1);

        System.out.print("Octets historiques (" + histLen + ") : ");
        for (byte b : historical) {
            System.out.printf("%02X ", b);
```

```
        }
        System.out.println();
    }
```

Exécutez le programme et observez la sortie console.

Questions : Partie A

Q1. Quel est l'octet TS affiché ? Quel est son rôle général dans un ATR (en une phrase) ?

Q2. Combien d'octets d'interface sont présents dans cet ATR ? Lesquels ?

Q3. Combien y a-t-il d'octets historiques ? Quels sont-ils (en hexadécimal) ?

Q4. Citez deux exemples d'informations que le fabricant pourrait mettre dans les octets historiques.

Q5. Expliquez en quelques lignes le rôle global de l'ATR dans la communication carte–lecteur.

Partie B : Modéliser le système de fichiers MF / DF / EF

Nous allons créer une modélisation objet de l'arborescence de fichiers d'une carte à puce (MF, DF, EF) et construire une carte SIM simplifiée en Java.

B.1. Création des classes de base

Créez la classe abstraite CardFile :

```
public abstract class CardFile {

    protected String fid;  // File ID (ex: "3F00", "7F10")
    protected String name; // Nom "humain" du fichier

    public CardFile(String fid, String name) {
        this.fid = fid;
        this.name = name;
    }

    public String getFid() {
        return fid;
    }

    public String getName() {
        return name;
    }

    /**
     * Affiche l'arborescence à partir de ce fichier.
     * Le paramètre indent permet de gérer l'indentation.
     */
    public abstract void printTree(String indent);
}
```

Créez l'énumération **EfStructure** :

```
public enum EfStructure {
    TRANSPARENT,
    LINEAR_FIXED,
    CYCLIC
}
```

Créez la classe **MasterFile** :

```
import java.util.ArrayList;
import java.util.List;

public class MasterFile extends CardFile {

    private final List<CardFile> children = new ArrayList<>();

    public MasterFile(String fid, String name) {
        super(fid, name);
    }

    public void addChild(CardFile file) {
        children.add(file);
    }

    @Override
    public void printTree(String indent) {
        System.out.println(indent + "[MF] " + name + " (FID=" + fid + ")");
        for (CardFile child : children) {
            child.printTree(indent + "  ");
        }
    }

    public CardFile findByFid(String targetFid) {
        if (this.fid.equalsIgnoreCase(targetFid)) {
            return this;
        }
        for (CardFile child : children) {
            CardFile res = findByFidRecursive(child, targetFid);
            if (res != null) return res;
        }
        return null;
    }

    private CardFile findByFidRecursive(CardFile file, String targetFid) {
        if (file.getFid().equalsIgnoreCase(targetFid)) {
            return file;
        }
        if (file instanceof DedicatedFile df) {
            for (CardFile c : df.getChildren()) {
                CardFile res = findByFidRecursive(c, targetFid);
                if (res != null) return res;
            }
        }
        return null;
```

```
    }
}
```

Créez la classe **DedicatedFile** :

```
import java.util.ArrayList;
import java.util.List;

public class DedicatedFile extends CardFile {

    private final List<CardFile> children = new ArrayList<>();

    public DedicatedFile(String fid, String name) {
        super(fid, name);
    }

    public void addChild(CardFile file) {
        children.add(file);
    }

    public List<CardFile> getChildren() {
        return children;
    }

    @Override
    public void printTree(String indent) {
        System.out.println(indent + "[DF] " + name + " (FID=" + fid + ")");
        for (CardFile child : children) {
            child.printTree(indent + "  ");
        }
    }
}
```

Créez la classe **ElementaryFile** :

```
public class ElementaryFile extends CardFile {

    private final EfStructure structure;

    public ElementaryFile(String fid, String name, EfStructure structure) {
        super(fid, name);
        this.structure = structure;
    }

    public EfStructure getStructure() {
        return structure;
    }

    @Override
    public void printTree(String indent) {
        System.out.println(indent + "[EF-" + structure + "] " + name + " (FID="
+ fid + ")");
    }
}
```

B.2. Construire une carte SIM simplifiée en Java

Créez maintenant la classe **SimCardFsDemo** avec la méthode main suivante :

```
public class SimCardFsDemo {

    public static void main(String[] args) {
        // Création du MF racine
        MasterFile mf = new MasterFile("3F00", "MF");

        // DF Télécom
        DedicatedFile dfTelecom = new DedicatedFile("7F10", "Telecom");

        // EF sous DF Telecom
        ElementaryFile efImsi     = new ElementaryFile("6F07", "IMSI",
EfStructure.TRANSPARENT);
        ElementaryFile efContacts = new ElementaryFile("6F3A", "Contacts",
EfStructure.LINEAR_FIXED);
        ElementaryFile efSms      = new ElementaryFile("6F3C", "SMS",
EfStructure.CYCLIC);

        dfTelecom.addChild(efImsi);
        dfTelecom.addChild(efContacts);
        dfTelecom.addChild(efSms);

        // On rattache DF Telecom au MF
        mf.addChild(dfTelecom);

        // Affichage de l'arborescence
        mf.printTree("");

        // Exemple de recherche par FID
        CardFile found = mf.findByFid("6F3C");
        if (found != null) {
            System.out.println("Fichier trouvé : " + found.getName() + " (FID="
+ found.getFid() + ")");
        } else {
            System.out.println("Fichier non trouvé.");
        }
    }
}
```

Exécutez le programme et observez l'affichage de l'arborescence.

Questions : Partie B

Q6. Recopiez l'arborescence affichée et expliquez la signification de chaque niveau (MF, DF Telecom, EF IMSI, EF Contacts, EF SMS).

Q7. Pourquoi IMSI est-il modélisé comme un EF transparent et non comme un EF linéaire ou cyclique ?

Q8. Pourquoi Contacts est-il modélisé comme un EF linéaire à enregistrements fixes ?

Q9. Pourquoi SMS est-il modélisé comme un EF cyclique ? Donnez un exemple concret de fonctionnement.

Q10. Proposez une autre application (carte bancaire, carte d'identité, etc.) et décrivez rapidement son organisation MF/DF/EF.