Sorbonne Université – License Informatique

LU3IN033 : Projet Analyseur de Protocoles Réseau ‘Offline’

Année 2020-2021

Fichier Read Me du projet

Binôme :

Koceila KIRECHE 28717834

Chems Eddine BENSAFIA 28717430

Description de la structure de notre code :

Langage utilisé : JAVA (jdk 11.0.2)

Bibliothèque graphique : Java FX 11

Patterns de conception utilisé : MVC

Notre code suit le patterns de conception **MVC,** nous séparer la vue du modèle du contrôleur :

Le modèle : notre modèle contient les classes qui représentent les couches, et pour suivre les recommandations de la bonne programmation orientée objets, nous avons favorisé la composition à l'héritage.

Toutes les classes obéissent à l'interface " IToFile " qui a une méthode, dont la signature est la suivante " public String fromDataToFile() ". Ce contrat permet de garantir à notre application que toutes les classes sont transformables en une description textuelle.

Concernant les classes des options IP, elles obéissent aussi à l'interface " IPOption " qui a les méthode suivantes

```
public String getType() : permet d'avoir le type

public String getLongeur() : permet de récupérer la
longueur

public String getValeur() : permet de récupérer la valeur

public String getName() : permet de récupérer le nom
textuel de l'option
```

Et vu que le modèle doit contenir les données et connaître aussi leurs définition, nous avons implanter dans le package " lecture " la classes Lecture qui permet de transformer un fichier '.txt' contenant les trames selon le format de capture WireShark en des dictionnaire (Map) de liste d'octets (qui sera traiter par classes métier du modèle) cette fonction permet de sélectionner les trames candidates (qui ne présente pas d'erreurs repérables directement à la lecture des données brutes) ce qui nous permet d'avoir une présélection des données.

En plus des données récupérées la classe Lecture nous donne accès à une liste d'erreurs rencontrées lors de la lecture et du décryptage du fichier, la définition de l'erreur se trouve dans le package lecture et elle a deux attributs clé qui sont « le numéro de ligne » et le « type de l'erreur » ce dernier peut prendre plusieurs valeurs comme ['caractère non hexadécimal, 'ligne incomplète'…] et cette liste d'erreur sera communiqué à la vue via le contrôleur.

Concernant la gestion des options, nous avons utilisé le design pattern Factory, donc la classe IP connais uniquement le contrat de l'interface et IPOption et c'est la méthode statique de la classe FactoryOptions qui se charge d'instancier la bonne classe.

Vue : nos vues sont séparées du code dans des fichier XML ce qui nous permet d'avoir une bonne séparation entre la vue, le modèle et le contrôleur, nous avons aussi séparé le style (colleurs, fonts, police, …) dans une fichier CSS, ceci nous permet d'avoir une vue passive et respecter les recommandations du pattern de conception MVC.

Le contrôleur : vu que nous utilisons la bibliothèque graphique Java Fx le contrôleur obéit à l'interface Initializable qui a une méthode :

```
public void initialize(URL url, ResourceBundle resourceBundle)
```

La bibliothèque Java Fx nous impose de séparer le contrôleur en plusieurs classes et chaque classe contrôlera une interface bien délimitée, la méthode " initialize " permet d'initialiser cette fenêtre et c'est grâce à cette dernière que le contrôleur communique les données du modèle à la vue.

Vu que nous avons plusieurs trames et que l'utilisateur peut afficher les détaille de ces dernières en cliquant sur l'une d'entre elles, nous avons une gestion d'état dans la classe main du programme qui permet de savoir quelle est la trame sélectionnée.

Petite description de la classe Lecture :

```
private void lire(File file) {
    String courantline;
    BufferedReader br = null;

    try {
        br = new BufferedReader(new FileReader(file));
        while ((courantline = saut(br)) != null) {
            this.lectureDeTrameCourant(courantline, br);
        }
        br.close();
    } catch (IOException e) {
        messageErreur(e.getMessage());
    }

}
```

La méthode principale de cette classe est la méthode " lire " qui lit le fichier passé en paramètre suivant la boucle qui a comme condition tant que la ligne retournée par la méthode saut est différente de Null.

La méthode saut permet d'ignorer et de sauter toutes les lignes qui n'ont pas un offset valide, et retourne la première ligne ayant un offset valide (qui débute par 0000).

Une fois cette ligne repérée, elle est passée à la méthode " lectureDeTrameCournat (ligne, br) " qui permet de lire la trame courante. Si elle ne détecte pas une erreur de

type (caractère non hexadécimal ou ligne incomplète) elle ajoute la trame à la structure Map avec comme clé la ligne ou elle commence dans le fichier lu.

Si une erreur est détectée elle est ajoutée à liste d'erreurs

```java
public class Error {
    private int numLigne;
    private String typeErreur;

    public Error(int numLigne, String typeErreur) {
        this.numLigne = numLigne;
        this.typeErreur = typeErreur;
    }

    @Override
    public String toString() {
        return "Erreur{" +
                "à la Ligne=" + numLigne +
                ", de type='" + typeErreur + '\'' +
                '}';
    }
}
```

La classe lecture nous permet de faire une présélection et d'avoir un certain nombre de trames candidates, la vérification de la taille de la trame se fait dans la classe IP en comparant entre le Total Length et la taille des données la trame, et ça, grâce à la méthode " verificationData " :

```java
private void verificationData(String trame[], String totalLength) throws DataNoValide {

    int tl = Integer.parseInt(totalLength, 16);
    if ((tl != (trame.length - 14))) {
        throw new DataNoValide("Donnée non Valide");

    }

}
```

Une exception de type " DataNoValide " sera levé dans le cas de la non validité des données, cette exception sera capturé par le contrôleur principale et une erreur sera ajoutée à la liste d'erreurs.