networkanalyzer

Binome


```
Amayas SADI  28717408
Hamid KOLLI  28717594
```


```
Video  youtube : #TODO
Github : https://github.com/Amayas29/networkanalyzer
Language : java
interface graphique : javafx (MVC) ou terminal
```

Une application d'analyse de trames ethernet

Description du projet

L'application Networkanalyzer prend en parametre un fichier de trames et l'analyse tout en decodant les trames.

> les protocoles supportés par notre application :

Ethernet, IP, ARP, ICMP, UDP, DNS, DHCP

> les options et types decodés :

IP Options : 0, 1, 7, 131, 137

DNS types : A, AAAA, CNAME, NS, MX

DHCP Options : 0, 1, 3, 6, 12, 28, 43, 50, 51, 53, 54, 55, 57, 58, 59, 60 ,61,255

DHCP messages types : les 10 premiers

Algorithme d'analyse

l'analyse se deroule en 2 phases

Premiere phase

Une phase ou l'application crée un autre fichier contenant l'ensemble des trames (qui commencent par l'offset 0 dans le fichier fourni) avec une liste de pattern qui permet de detecter les lignes contenant des erreurs, le patern est sous format [nombre de ligne, l'indice du premier caractere de la ligne]

Cette premiere phase permet d'echappé les valeurs textuelles, les sauts de ligne et verifier l'offset s'il est valide

l'algorithme en code :

```
while ((line = bufferedReader.readLine()) != null) {

                data = line.split(" ");
                indexLine++;

                if (data.length == 0)
                    continue;

                if (checkOffset(data[0], "0", 0)) {
                    if (sb.length() != 0)
                        bufferedWriter.write(indexs.toString()
                                .concat(sb.toString().concat(" ").concat(String.valueOf(index * 3)).concat("\n")));

                    lengthLine = 0;
                    oldOffset = "0";
                    sb.setLength(0);
                    indexs.setLength(0);
                }

                if (!checkOffset(oldOffset, data[0], lengthLine))
                    continue;

                oldOffset = data[0];
                indexs.append(addPattern(indexLine, index * 2 + index).concat(" "));

                lengthLine = 0;

                for (int i = 1; i < data.length; i++) {

                    if (data[i].isBlank())
                        continue;

                    if (!checkByte(data[i]))
                        break;

                    sb.append(data[i].concat(" "));
                    lengthLine++;
                    index++;
                }
            }

            if (sb.length() != 0)
                bufferedWriter.write(
                        indexs.toString().concat(sb.toString().concat(" ").concat(String.valueOf(index * 3)).concat("\n")));

        } catch (

        IOException e) {
            throw new NetworkAnalyzerFileErrorsException(e.getMessage());
        } finally {
            if (bufferedWriter != null)
                try {
                    bufferedWriter.close();
                } catch (IOException e) {
                    throw new NetworkAnalyzerFileErrorsException(e.getMessage());
                }
        }
```

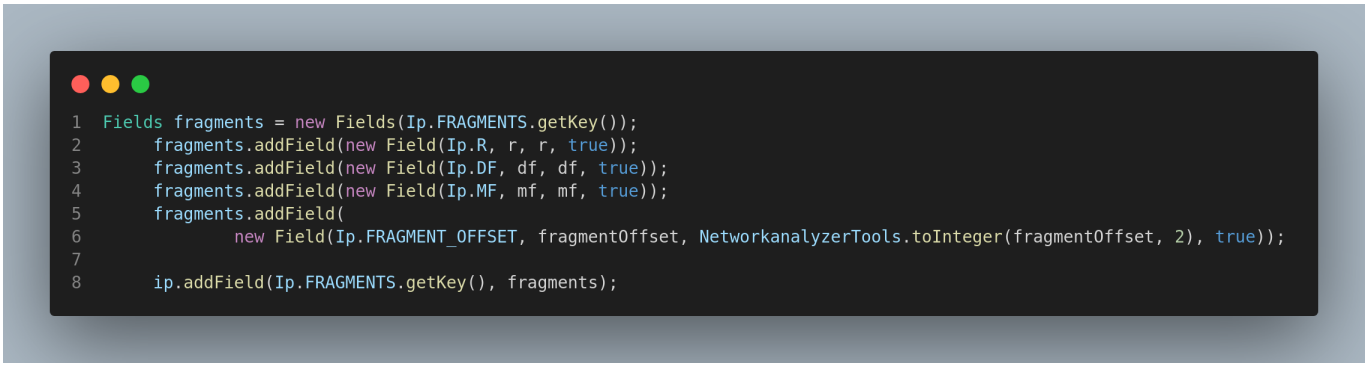
Champs statiques des protocoles

Pour les champs statiques, on les a analysé directement en extrant le nombre d'octets exacts et le decoder en utilisant les methodes statiques de la classe **NetworkanalyzerTools** et en testant leurs valeurs (par exemple l'adresse mac source ne doit pas etre un broadcast)

un exemple d'analyse des champs statiques

```java
public void visit(Ethernet ethernet) throws NetworkAnalyzerException {
        header = getHeader(42).trim();

        String destMacAddress = parseField(Ethernet.DEST_ADDRESS);
        incIndex(Ethernet.DEST_ADDRESS);

        String srcMacAddress = parseField(Ethernet.SRC_ADDRESS);

        if (srcMacAddress.equals("FF FF FF FF FF FF")) {

            int lineError = getLine();
            moveToNextFrameIndex();
            throw new NetworkanalyzerParseErrorException(lineError,
                    "The source MAC address must not be a broadcast address");
        }

        if (destMacAddress.equals(srcMacAddress) && !destMacAddress.equals("00 00 00 00 00 00")) {
            int lineError = getLine();
            moveToNextFrameIndex();
            throw new NetworkanalyzerParseErrorException(lineError, "Mac addresses are equal");
        }

        incIndex(Ethernet.SRC_ADDRESS);

        String rdType = parseField(Ethernet.TYPE);

        IField type = null;
        ILayerNetwork layer = null;
        boolean isData = false;
        int indexWarning = 0;
        switch (rdType) {

        case Ethernet.IPV4: {
            layer = new Ip();
            type = new Field(Ethernet.TYPE, rdType, layer.getName());
            break;
        }

        case Ethernet.ARP: {
            layer = new Arp();
            type = new Field(Ethernet.TYPE, rdType, layer.getName());
            break;
        }

        case Ethernet.IPV6: {
            isData = true;
            indexWarning = getLine();
            type = new Field(Ethernet.TYPE, rdType, "IPV6");
            break;
        }

        default:

            int lineError = getLine();
            moveToNextFrameIndex();
            throw new NetworkanalyzerParseErrorException(lineError, "Unexpected value of the ethernet type field");
        }

        incIndex(Ethernet.TYPE);

        Field dest = new Field(Ethernet.DEST_ADDRESS, destMacAddress,
                destMacAddress.equals("FF FF FF FF FF FF") ? "broadcast" : destMacAddress.replace(" ", ":"));

        Field src = new Field(Ethernet.SRC_ADDRESS, srcMacAddress, srcMacAddress.replace(" ", ":"));

        ethernet.addField(Ethernet.SRC_ADDRESS.getKey(), src);
        ethernet.addField(Ethernet.DEST_ADDRESS.getKey(), dest);
        ethernet.addField(Ethernet.TYPE.getKey(), type);

        if (!isData) {
            ethernet.setIncluded(layer);
            layer.accept(this);

            return;
        }

        moveToNextFrameIndex();
        Entry<String, Integer> dataEntry = Ethernet.DATA.setValue(line.split(" ").length * 8);
        ethernet.addField(dataEntry.getKey(), new Field(dataEntry, line, line));
        throw new NetworkanalyzerParseWarningException(indexWarning, " IPV6 is not supported");
    }
```

un exemple d'analyse des flags



```
Fields fragments = new Fields(Ip.FRAGMENTS.getKey());
      fragments.addField(new Field(Ip.R, r, r, true));
      fragments.addField(new Field(Ip.DF, df, df, true));
      fragments.addField(new Field(Ip.MF, mf, mf, true));
      fragments.addField(
              new Field(Ip.FRAGMENT_OFFSET, fragmentOffset, NetworkanalyzerTools.toInteger(fragmentOffset, 2), true));

      ip.addField(Ip.FRAGMENTS.getKey(), fragments);
```

Champs dynamiques des protocoles

pour les champs dynamiques on trouve 2 cas, soit pour les options de DHCP et IP, ou pour la decompression de DNS

pour les options, on utilise les methodes statiques de OptionBuilder qui sont **buildIpOptions(String)** et **buildDhcpOptions(String)** ou on parcoure la liste des octets qui sont passés en parametre des methodes et on teste le champs type d'option, si c'est une option que notre analyser supporte on la decode selon sa taille sinon on ajoute dans les octets en brut

un exemple d'analyse d'option

```java

public static IField buildIpOptions(String header) throws NetworkAnalyzerException {

        Fields options = new Fields(Ip.OPTIONS.getKey(), true);

        String data[] = header.trim().split(" ");
        for (int i = 0; i < data.length;) {

            String type = data[i++];
            int typeDecoded = Integer.parseInt(type, 16);

            Entry<String, Integer> typeEnty = IpOptions.getEntryByCode(typeDecoded);
            Entry<String, Integer> t = new Entry<>("Option " + typeEnty.getKey(), 8);
            if (typeDecoded == 68)
                continue;

            if (typeDecoded == 1 || typeDecoded == 0) {
                options.addField(new Field(t, type, String.valueOf(typeDecoded)));
                continue;
            }

            Fields option = new Fields(String.format("%s %d", typeEnty.getKey(), typeDecoded), true);
            option.addField(new Field(new Entry<>("Type", 8), type, String.valueOf(typeDecoded)));

            String len = data[i++];
            int lenDecoded = Integer.parseInt(len, 16);
            option.addField(new Field(new Entry<>("Length", 8), len, String.valueOf(lenDecoded)));

            if (typeDecoded == 7) {
                String ptr = data[i++];
                option.addField(new Field(new Entry<>("Pointer", 8), ptr, String.valueOf(Integer.parseInt(ptr, 16))));
                lenDecoded -= 1;
            }

            lenDecoded -= 2;
            Fields fieldsAdresses = new Fields("Address", true);

            for (int j = 3, k = 1; j < lenDecoded; j += 4, k += 1) {
                String ips = String.format("%s %s %s %s", data[j], data[j + 1], data[j + 2], data[j + 3]);
                fieldsAdresses.addField(
                        new Field(new Entry<>("address " + k, 32), ips, NetworkanalyzerTools.decodeAddressIp(ips)));
                i += 4;

            }

            option.addField(fieldsAdresses);
            options.addField(option);

        }

        return options;
    }

```

Pour les champs variables de DNS (exemple **Questions**), on a bouclé le nombre lu dans le champs statique correspondant (exemple **Questions Number**) pour recuperer tout les champs variables

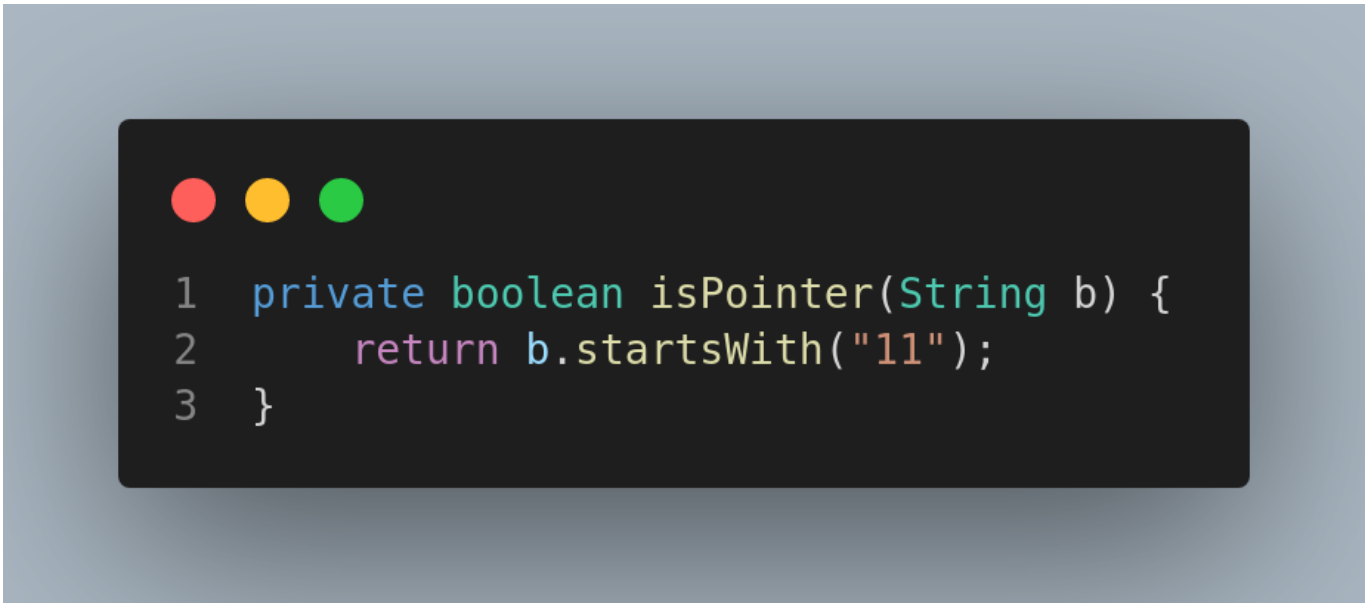
la fonction utilisé

```java
private int parseDnsNames(int number, String data[], int curr, Fields container, boolean isQuestions) {

        for (int j = 0; j < number; j++) {
            StringBuilder sb = new StringBuilder();

            Fields item = new Fields(String.format("%s %d", "N° ", j), true);
            container.addField(item);

            curr = getDnsName(data, curr, item);

            String type = String.format("%s %s", data[curr], data[curr + 1]);
            curr += 2;

            String cls = String.format("%s %s", data[curr], data[curr + 1]);
            curr += 2;

```

```
            Field t = new Field(new Entry<String, Integer>(isQuestions ? "QTYPE" : "TYPE", 16), type,
                    DnsDecoder.getType(Integer.parseInt(type.replace(" ", ""), 16)).getKey());

            Field c = new Field(new Entry<String, Integer>(isQuestions ? "QCLASS" : "CLASS", 16), cls,
                    DnsDecoder.getClassName(Integer.parseInt(cls.replace(" ", ""), 16)));

            item.addField(t);
            item.addField(c);

            String rdataLength = "0";
            int numberData = 0;

            if (!isQuestions) {
                String ttl = String.format("%s %s %s %s", data[curr], data[curr + 1], data[curr + 2], data[curr + 3]);
                curr += 4;

                rdataLength = String.format("%s %s", data[curr], data[curr + 1]);
                curr += 2;

                Field tl = new Field(new Entry<String, Integer>("TTL", 32), ttl, NetworkanalyzerTools.toInteger(ttl));
                numberData = Integer.parseInt(NetworkanalyzerTools.toInteger(rdataLength));

                Field rData = new Field(new Entry<String, Integer>("RDATA LENGTH", 16), rdataLength, numberData + "");

                item.addField(tl);
                item.addField(rData);
            }

            if (!isQuestions) {

                String decType = "0x" + type.replace(" ", "").toLowerCase();
                String ipAddress;
                Field ip;

                if (decType.equals("0x0001")) {
                    ipAddress = String.format("%s %s %s %s", data[curr], data[curr + 1], data[curr + 2],
                            data[curr + 3]);
                    curr += 4;
                    ip = new Field(new Entry<String, Integer>("IP ADDRESS", 32), ipAddress,
                            NetworkanalyzerTools.decodeAddressIp(ipAddress));
                    item.addField(ip);
                }

                else if (decType.equals("0x001c")) {
                    ipAddress = String.format("%s %s %s %s %s %s %s %s", data[curr], data[curr + 1], data[curr + 2],
                            data[curr + 3], data[curr + 4], data[curr + 5], data[curr + 6], data[curr + 7]);

                    String decIp = String.format("%s%s:%s%s:%s%s:%s%s:%s%s:%s%s:%s%s:%s%s", data[curr], data[curr + 1],
                            data[curr + 2], data[curr + 3], data[curr + 4], data[curr + 5], data[curr + 6],
                            data[curr + 7]);

                    curr += 6;

                    ip = new Field(new Entry<String, Integer>("IP ADDRESS", 32), ipAddress, decIp);
                    item.addField(ip);

                }

                else if (decType.equals("0x0005") || decType.equals("0x0002") || decType.equals("0x000F"))
                    curr = getDnsName(data, curr, item);

                else {

                    int l = Integer.parseInt(rdataLength.replace(" ", ""), 16);

                    sb = new StringBuilder();
                    for (int k = 0; k < l; k++)
                        sb.append(data[curr++]).append(" ");

                    item.addField(new Field(new Entry<String, Integer>("Data", l), sb.toString(), "Uknown data"));
                }
            }
        }

        return curr;
    }
```

et pour decompresser les noms dans DNS, on a bouclé jusqu'a ce qu'on trouve l'octet **00** qui definit la fin du nom, et en testant si on doit sauter vers un label (si le champs qui remplace la taille commence par les bits **11** et les 14 autre bits correspondent au label)

pour tester si le premier octet du champs nom est un label, on a utilisé cette methode



```java
private boolean isPointer(String b) {
    return b.startsWith("11");
}
```

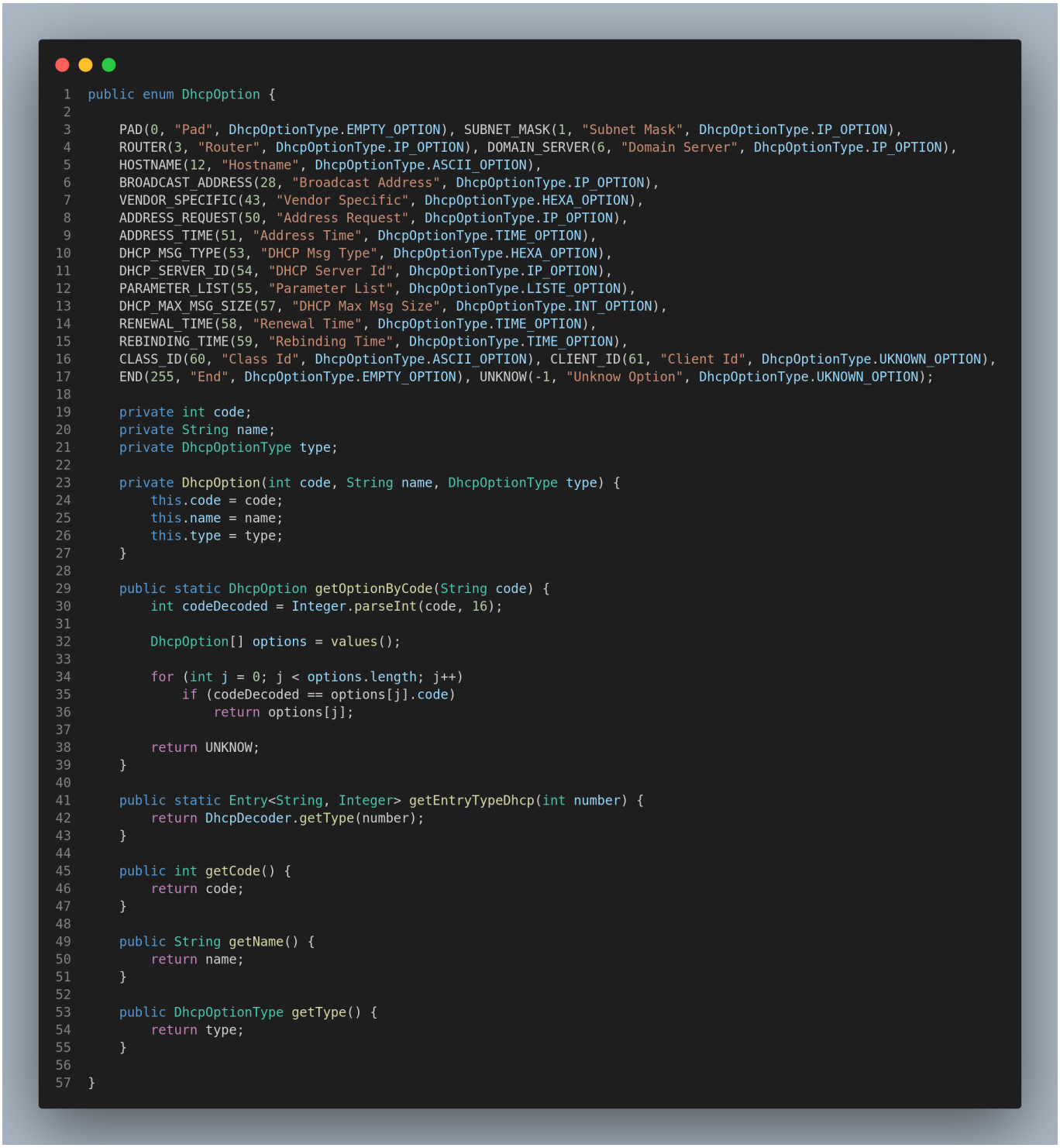
et la fonction qui decompresse le name:

```java
private int getDnsName(String data[], int curr, Fields fields) {
        boolean notJumped = true;
        int i = curr;
        StringBuilder sbV = new StringBuilder();
        StringBuilder sbN = new StringBuilder();
        while (!data[i].equals("00")) {
            if (isPointer(NetworkanalyzerTools.hexToBinEncoded(data[i]))) {
                if (notJumped) {
                    sbV.append(String.format("%s %s ", data[i], data[i + 1]));
                    notJumped = false;
                    curr += 2;
                }
                i = findName(data, i);
            }
            else {
                int len = Integer.parseInt(data[i], 16);
                sbN.append(data[i]).append(" ");
                if (notJumped) {
                    sbV.append(data[i]).append(" ");
                    curr++;
                }
                i++;
                int k = i;
                for (int j = 0; j < len; j++) {

                    if (notJumped) {
                        sbV.append(data[k + j]).append(" ");
                        curr++;
                    }
                    sbN.append(data[k + j]).append(" ");
                    i++;
                }
            }
        }
        if (notJumped) {
            sbV.append("00");
            curr++;
        }

        String name = sbN.toString().strip();
        String value = sbV.toString().strip();
        String decoded = getDnsNameDecoded(name);

        Field f = new Field(new Entry<>("NAME", 0), value, decoded, name);
        fields.addField(f);

        return curr;
    }
```

Recuperation dynamique des noms des options et des types

Au lieu d'utiliser des **if else** a chaque fois pour decoder le code des options , on a utilisé des **enum** et la methode **getValue** pour recuperer dynamiquement les noms des options par exmeple (le code 7 dans les options de IP correspant à **RECORD ROOT**)

Exemple : DhcpOption



```java
public enum DhcpOption {

    PAD(0, "Pad", DhcpOptionType.EMPTY_OPTION), SUBNET_MASK(1, "Subnet Mask", DhcpOptionType.IP_OPTION),
    ROUTER(3, "Router", DhcpOptionType.IP_OPTION), DOMAIN_SERVER(6, "Domain Server", DhcpOptionType.IP_OPTION),
    HOSTNAME(12, "Hostname", DhcpOptionType.ASCII_OPTION),
    BROADCAST_ADDRESS(28, "Broadcast Address", DhcpOptionType.IP_OPTION),
    VENDOR_SPECIFIC(43, "Vendor Specific", DhcpOptionType.HEXA_OPTION),
    ADDRESS_REQUEST(50, "Address Request", DhcpOptionType.IP_OPTION),
    ADDRESS_TIME(51, "Address Time", DhcpOptionType.TIME_OPTION),
    DHCP_MSG_TYPE(53, "DHCP Msg Type", DhcpOptionType.HEXA_OPTION),
    DHCP_SERVER_ID(54, "DHCP Server Id", DhcpOptionType.IP_OPTION),
    PARAMETER_LIST(55, "Parameter List", DhcpOptionType.LISTE_OPTION),
    DHCP_MAX_MSG_SIZE(57, "DHCP Max Msg Size", DhcpOptionType.INT_OPTION),
    RENEWAL_TIME(58, "Renewal Time", DhcpOptionType.TIME_OPTION),
    REBINDING_TIME(59, "Rebinding Time", DhcpOptionType.TIME_OPTION),
    CLASS_ID(60, "Class Id", DhcpOptionType.ASCII_OPTION), CLIENT_ID(61, "Client Id", DhcpOptionType.UKNOWN_OPTION),
    END(255, "End", DhcpOptionType.EMPTY_OPTION), UNKNOW(-1, "Unknow Option", DhcpOptionType.UKNOWN_OPTION);

    private int code;
    private String name;
    private DhcpOptionType type;

    private DhcpOption(int code, String name, DhcpOptionType type) {
        this.code = code;
        this.name = name;
        this.type = type;
    }

    public static DhcpOption getOptionByCode(String code) {
        int codeDecoded = Integer.parseInt(code, 16);

        DhcpOption[] options = values();

        for (int j = 0; j < options.length; j++)
            if (codeDecoded == options[j].code)
                return options[j];

        return UNKNOW;
    }

    public static Entry<String, Integer> getEntryTypeDhcp(int number) {
        return DhcpDecoder.getType(number);
    }

    public int getCode() {
        return code;
    }

    public String getName() {
        return name;
    }

    public DhcpOptionType getType() {
        return type;
    }

}
```

On a utilisé le meme mecanisme pour les differents types, sauf qu'on a utilisé une table de hachage au lieu d'une enum (moins couteuse (O(1))), donc on a utilisé des classes **Decoder**

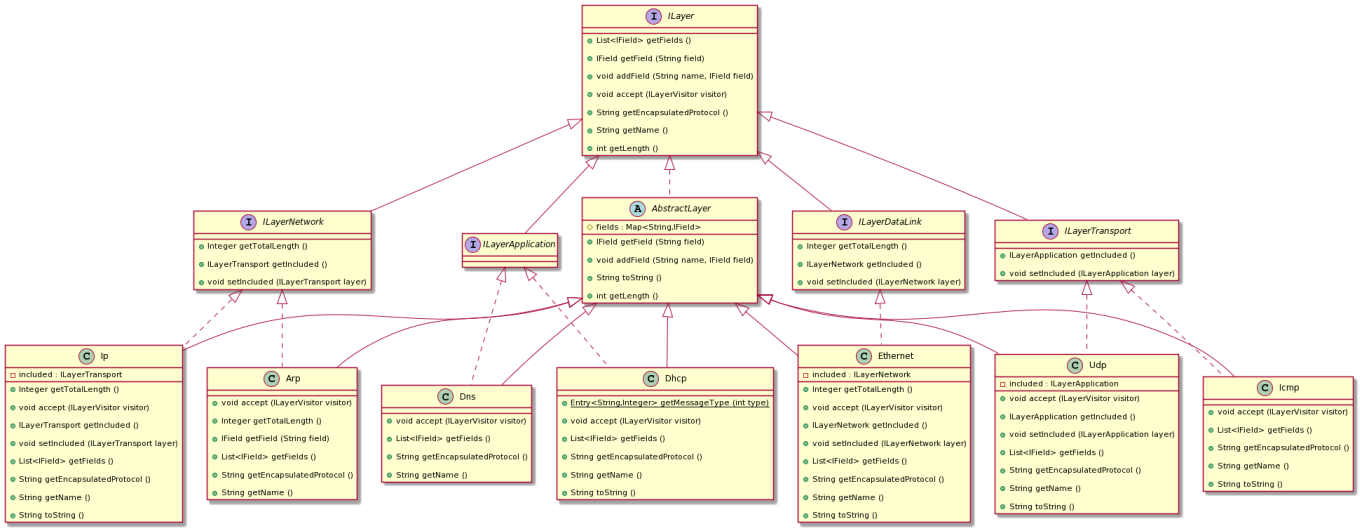
Exemple : DhcpDecoder

```java
public class DhcpDecoder extends Decoder {

    static {

        put(1, "Discover");
        put(2, "Offer");
        put(3, "Request");
        put(4, "Decline");
        put(5, "ACK");
        put(6, "NAK");
        put(7, "Release");
        put(8, "Inform");
        put(9, "Force Renew");
        put(10, "Lease query");
        put(-1, "Unknow");

    }

    public static Entry<String,Integer> getType(int i){
        return Decoder.getType(i);
    }


}
```

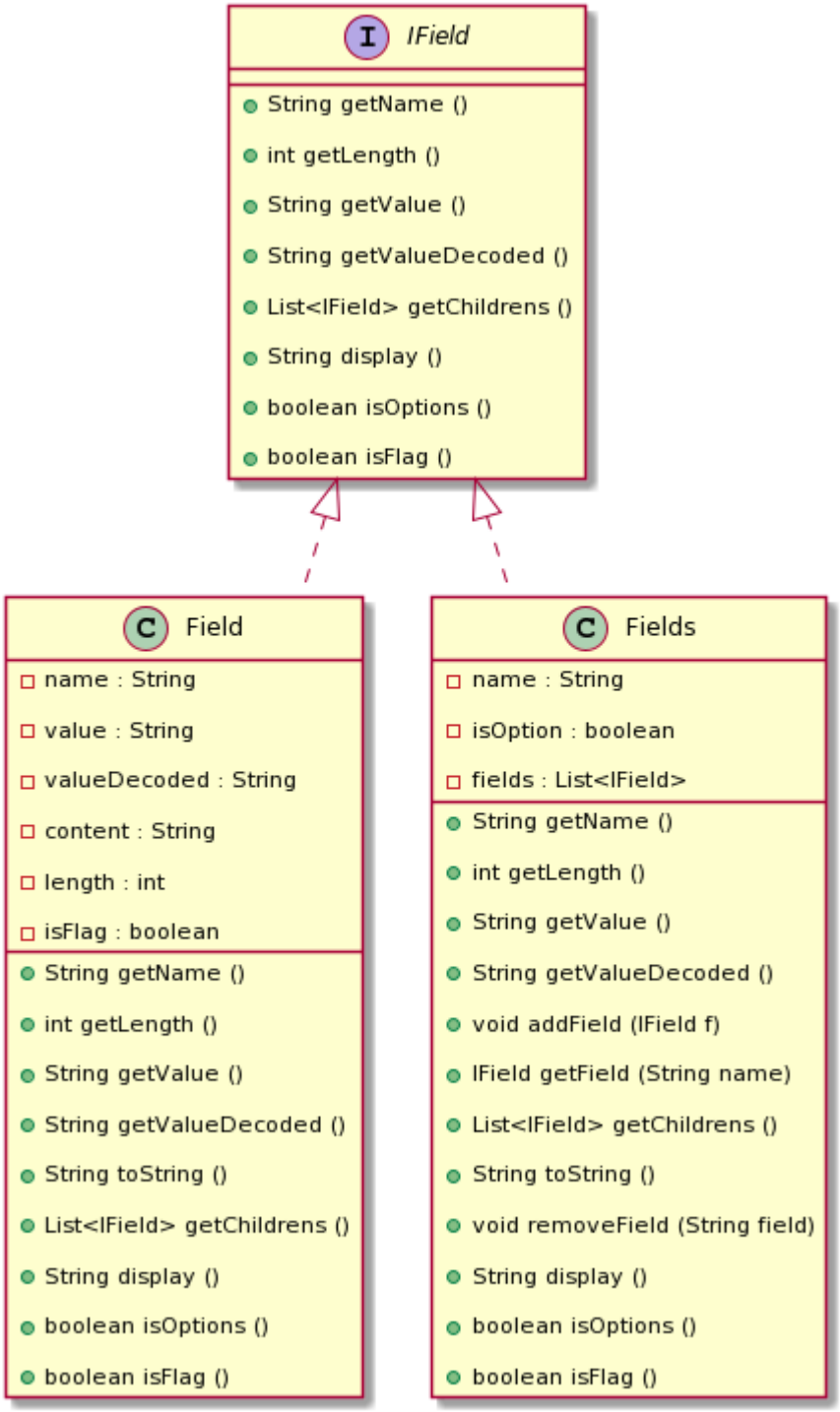
Structure du projet

Le Diagramme UML

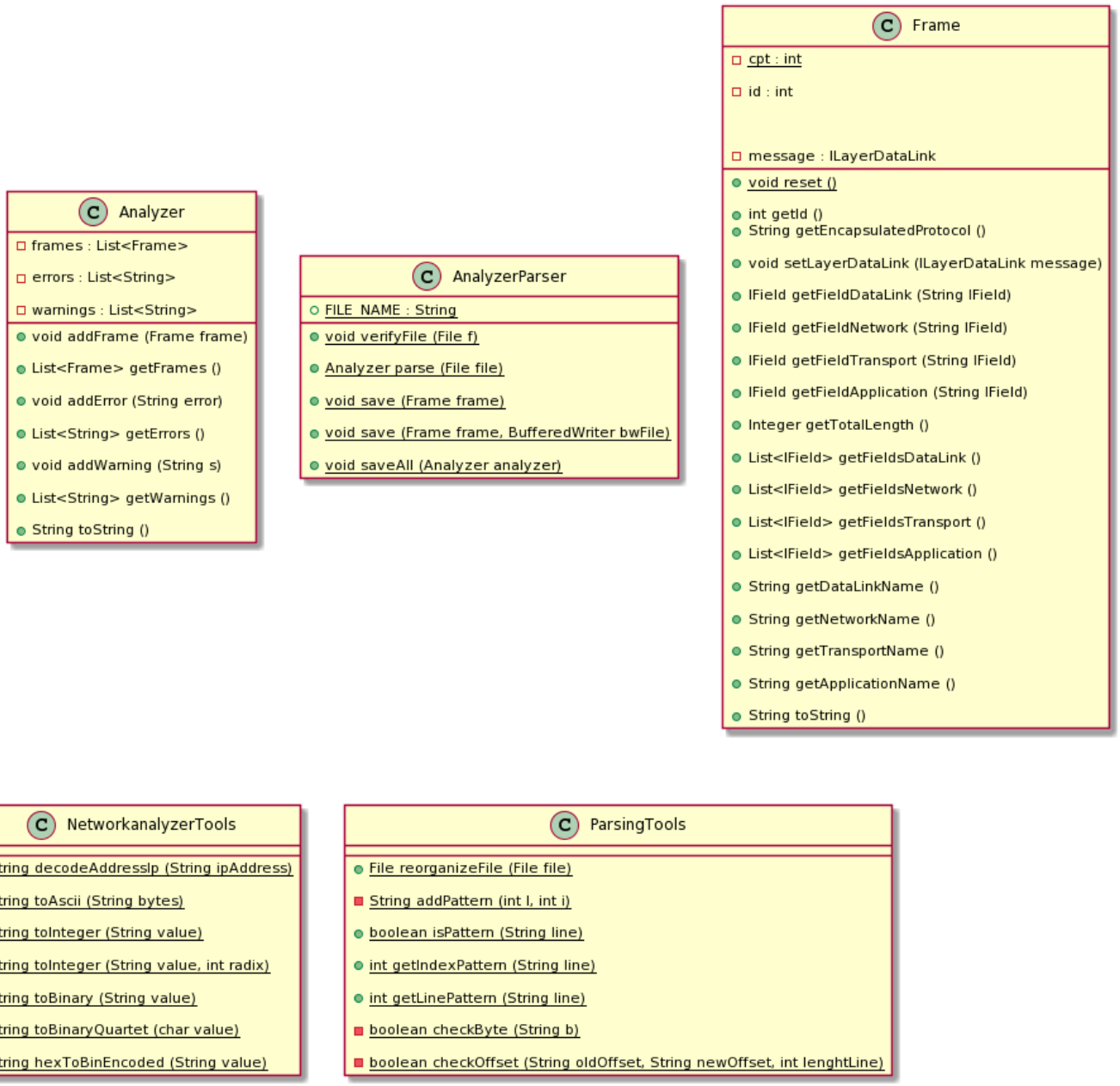
- Les couches sont representés par des interface implementant l'interface **Layer**, et chaque protocole est representé par une classe qui implemente sa couche (par exemple la classe **Ethernet** implemente **ILayerDataLink**) et qui herite **AbstractLayer** qui est une classe qui factorise une table de hachage et ses methodes, la table de hachage contient les champs du protocole avec comme clé le nom du champs. Les protocoles qui ont un autre protocole a encapsulé, contiennent un attribut **include** (Exemple : **Ip** contient un included de type **ILayerTransport**).



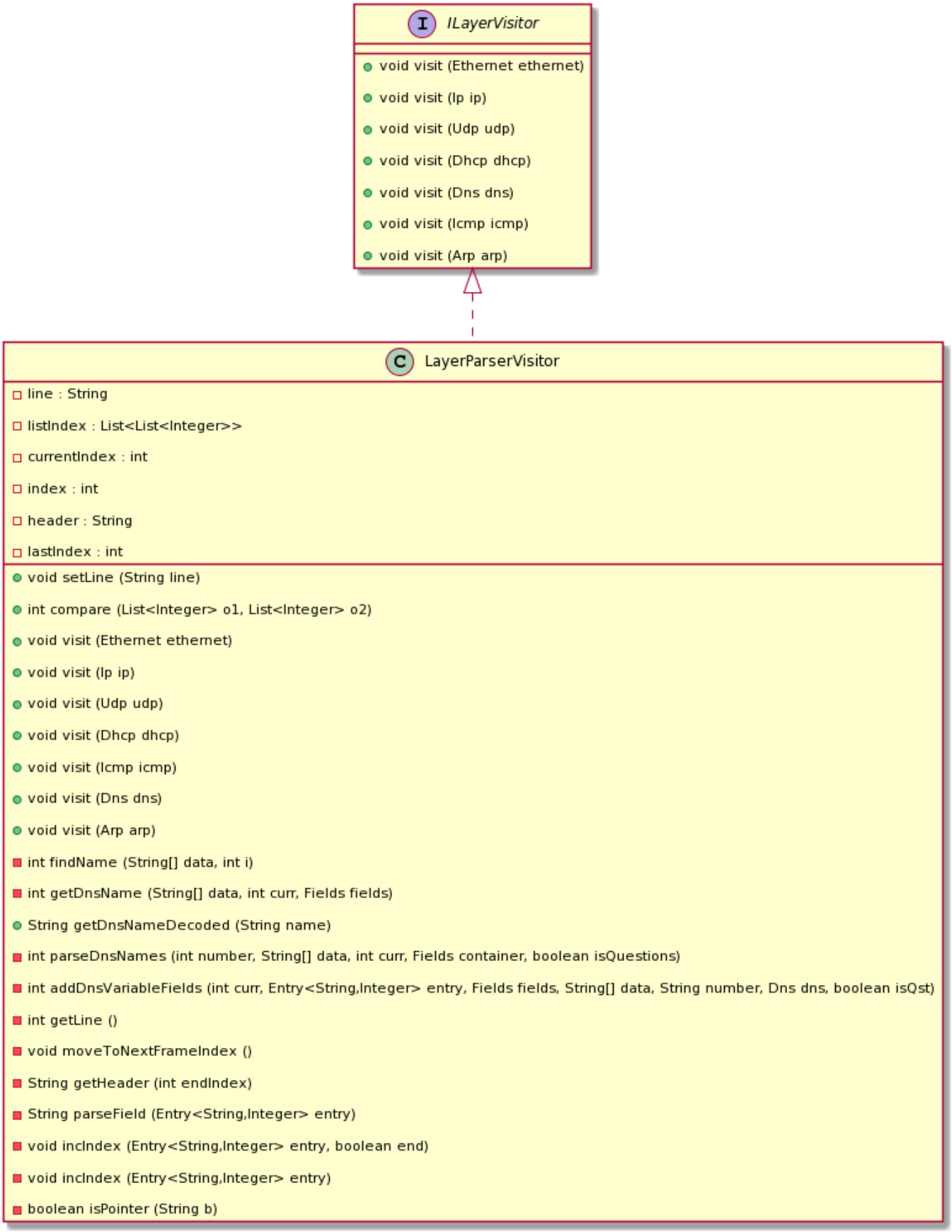
- Les champs sont representé par l'interface **IField**, les champs sont soit des champs simples donc des **Field** ou des conteneurs d'autre champs comme les flags ou les options donc des **Fields**



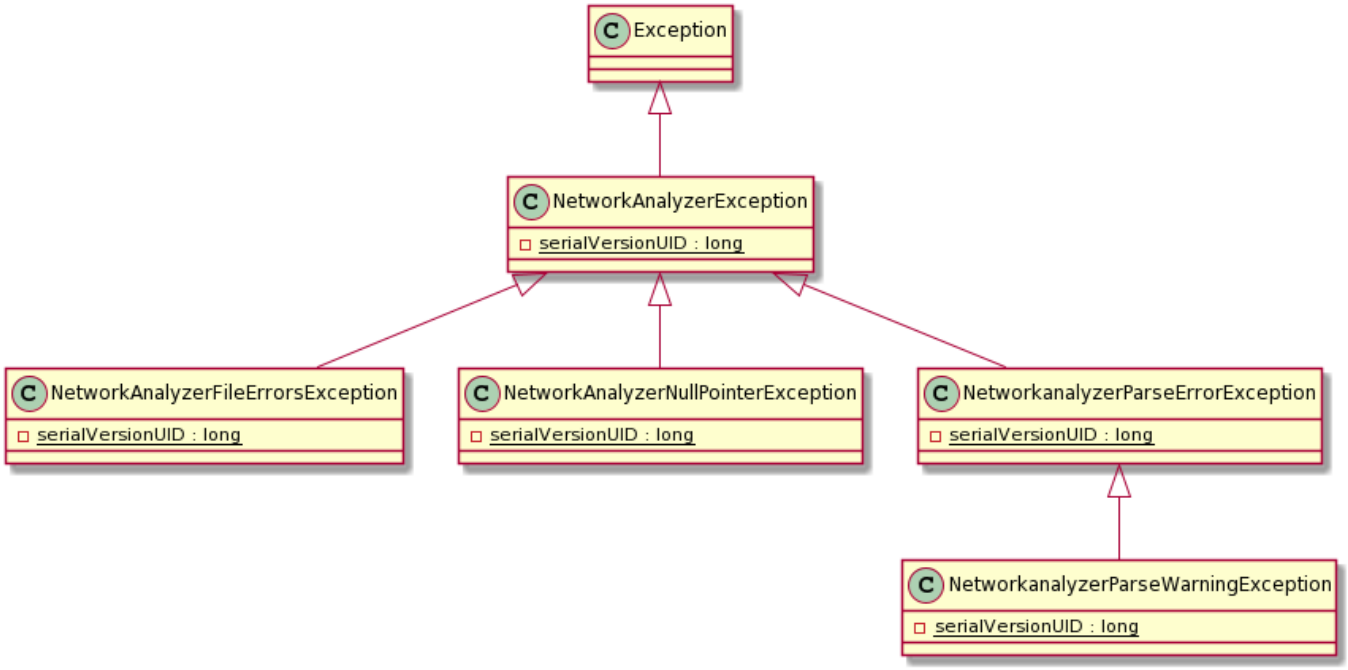
- Une trame est representé par la classe **Frame**, elle contient un seul attribut qui est de type **ILayerDataLink**, dans notre application on peut avoir que ethernet, et cette attribut encapsule toutes les autres couches
- Notre projet utilise une classe principale qui est **Analyzer** qui contient toutes les trames (list de **Frame**) et toutes les erreurs et les warnings de notre application.
- La methode **Analyzer parse(File)** de la classe **AnalyzerParser** permet de faire les deux phase de l'analyse, d'abord elle fait appel a la methode statique **File reorganizeFile(File)** de la classe **ParsingTools** qui est une classe qui contient que des methodes statiques que l'application utilise pour faire la premiere phase du parse. Ensuite **AnalyzerParser** lance la deuxieme phase du parse sur le nouveau fichier generé par **File reorganizeFile(File)** en applant la methode **void accept(ILayerVisitor)** (**DESIGN PATERN Visitor**)de l'objet du type **Ethernet**.



- L'application utilise le design pattern visitor pour faire la deuxieme phase de l'analyse, elle utilise une interface **ILayerVisitor** qui a des methodes **void visit(-)** avec des arguments des types des protocole, et la classe **LayerVisitor** implement cette interface et permet d'analyser dans chaque methode **visit** le protocole passée en parametre (Exemple **void visit(Ip)** permet d'analyser l'entête IP et lancer l'analyse du transport en utilisant la methode **accept(ILayerVisitor)** (qui est une methode definie dans chaque protocole et dans **ILayer**))



- Notre application utilise sa propre exception **NetworkanalyzerException**, et toutes les autres exceptions utilisés heritent de cette exception ou si on throw une autre exception propre a java on la capte et apres on throw une des **NetworkanalyzerException**


Exception
NetworkAnalyzerException
serialVersionUID : long
NetworkAnalyzerFileErrorsException
serialVersionUID : long
NetworkAnalyzerNullPointerException
serialVersionUID : long
NetworkanalyzerParseErrorException
serialVersionUID : long
NetworkanalyzerParseWarningException
serialVersionUID : long