TP N °4 JAVA M2207 Consolidation des bases de la programmation AAA Vendredi 12 février 2019

Exercice 1

Q1)

```java
import java.net.*;
import java.io.*;
public class URL2Fichier {
    /* Méthode principale */
    public static void main(String[] args) {
        final int TAILLE_TAMPON = 4096;
        URL uneUrl ;
        InputStream fluxE=null ;
        FileOutputStream fluxS=null ;
        BufferedOutputStream donneesEcrites = null;
        BufferedInputStream donneesLues = null;
        int nbLus=0 ;
        byte [] tampon ;
        //tampon de lecture et d'écriture
        if (args.length != 2) {
            System.err.println(" Le programme requiert 2 arguments ");
            System.err.println("l'URL du fichier à copier");
            System.err.println("le nom du fichier destination");
            System.exit(1) ;
        }
        try {
            // Ouverture de l'URL pour une lecture
            uneUrl = new URL(args[0]);
            //Convertir l'URL en InputStream
            fluxE=uneUrl.openStream() ;
            //Construction du flux de type BufferedInputStream
            donneesLues = new BufferedInputStream(fluxE);
            // construction d'un FileOutputStream.
            fluxS = new FileOutputStream(args[1]);
            // construction d'un BufferedOutputStream
            donneesEcrites = new BufferedOutputStream(fluxS);
            System.out.println("Connextion établie") ;
            tampon=new byte[TAILLE_TAMPON] ;
            do{
                nbLus=donneesLues.read(tampon) ;
                if (nbLus !=-1) {
                    donneesEcrites.write(tampon,0,nbLus) ;}
            }while(nbLus !=-1) ;
            donneesEcrites.close() ; //écriture de la fin du fichier
            System.out.println("Fin d'écriture") ;
        }
        catch (MalformedURLException excp) {
            System.out.println (args[0] + " : impossible de traiter cette URL");
            System.out.println (excp);
        }
        catch (IOException excp) {
            System.out.println ("Erreur : " + excp);
        }
    }
}
```

Explication :

D'abord on importe les bibliothèques, ensuite la classe URL2Fichier est créé.

Le tampon (*mémoire intermédiaire*) est une constante (d'où le terme final) pour 4096 octets.

URL uneUrl permet que lorsque l'on écrit uneUrl c'est comme si on écrivait toujours URL uneUrl, les variables sont déclarés de manière (null= *indéfinie*), les flux d'octets Stream et Buffered et le tampon

Puis la boucle if permet d'afficher des messages et d'obliger l'utilisateur à rentrer 2 arguments et d'arrêter le programme avec System.exit(1)

Le try essaye l'ouverture des URL, flux, construction des fichiers de destination ainsi dès que toute cette série d'initialisation est terminé alors il affiche la connexion est établie s'il y a une erreur alors un catch a lieu. (URL invalide, Fichier impossible à créer)

Le 1er bloc catch sert à indiquer si l'URL est invalide avec le message d'erreur.

Le 2ème bloc catch sert à indiquer si une erreur survient avant la fin d'écriture voire même sur le fichier d'entrée.

La boucle do permet de traiter le nombre de lignes à lire par conséquent tant qu'il y a des lignes à lire chaque ligne sont copiées du fichier d'entrée vers le fichier de destination. (*Si nbLus vaut -1 cela signifie qu'il n'y a plus rien à lire selon la méthode read)*

Les buffers et tampon permettent d'exploiter la robustesse du langage de programmation JAVA afin d'optimiser et de traiter ce programme.

Explications approfondies :

fluxE = url uneURL.openStream() : permet la conversion de l'URL en InputStream

openStream() ouvre le flux d'entrée en flux d'octet

BufferedInputStream et Output permettent la bufferisation ainsi on construit les flux

donneesEcrites.write(tampon,0,nbLus) le tampon est la source il récupère avec le rang 0, en fonction de nbLus. Donc : il copie vers le fichier de destination donneesEcrites

donneesEcrites.close() ferme le fichier de destination

Donc : Ce programme permet en d'autres termes de mettre le site en mode hors-connexion, c'est un sniffeur de site.

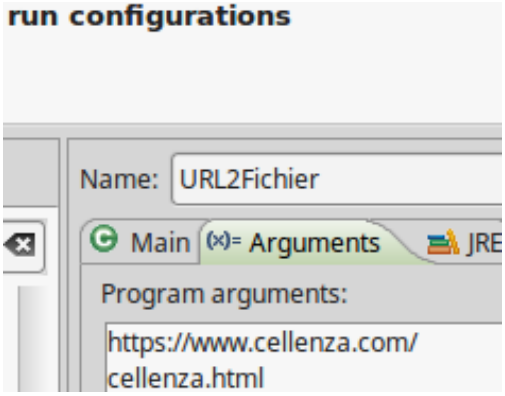
Résultat code original :

```
 Le programme requiert 2 arguments
l'URL du fichier à copier
le nom du fichier destination
```

Ce programme a besoin de 2 arguments par conséquent il ne marche pas sans arguments ou avec trop d'arguments. L'argument 0 est l'URL soit le 1er argument. L'argument 1 est le fichier de destination de préférence avec l'extension HTML ou txt soit le 2ème argument.

Une fois avoir rentré les arguments dans run -> run configurations , on a ceci :

NB : Les arguments peuvent être espacé ou avec un retour à la ligne




```
Connexion établie
Fin d'écriture
```

On peut d'ailleurs voire le fichier :

Autre résultat: On constate qu'avec le site u-pec on n'a pas récupéré les images, vidéos par conséquent on n'a que le contenu HTML

Les résultats d'erreurs :

Si l'URL n'existe pas alors on a UnknownHostException par java.net

```
Erreur : java.net.UnknownHostException: www.cellenzans.fr
```

On peut spécifier le protocole avec :80 (pour le protocole http) ou même avec un autre numéro de port

```
Connexion établie à http://www.perdu.com:80
Fin d'écriture dans myperdu.html
```

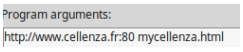
En cas d'erreur, dossier protégé, des répertoires inexistants (ici) une IOException est généré :

```
Erreur : java.io.FileNotFoundException: u/pecIOException (No such file or directory)
```

Ou même si l'on ne spécifie pas le fichier après le port ou dans l'URL :

```
Erreur : java.io.FileNotFoundException: http://www.cellenza.fr:80
```

Avec comme arguments :



En cas de protocole inconnu ce programme nous donne aussi la partie du protocole inconnue

```
htttps://www.cellenza.com/ : impossible de traiter cette URL
java.net.MalformedURLException: unknown protocol: htttps
```

Il indique qu'il n'y a pas de protocole pour cette URL

```
www.u-pec.fr : impossible de traiter cette URL
java.net.MalformedURLException: no protocol: www.u-pec.fr
```

Java.net.ConnectException peut être provoqué en cas d'impossibilité à copier le contenu de la page HTML ou si la connexion est bloquée par un pare-feu ou si la connexion est bloquée par des proxys…

Alors il y a délai d'expiration de la connexion

```
Erreur : java.net.ConnectException: Connection timed out: connect
```

Résultat avec un programme amélioré :

```
Connexion établie à http://www.u-pec.fr
Fin d'écriture dans upecdest.html
```

Pour obtenir un résultat plus pratique à identifier j'ai ajouté une concaténation des arguments

Exercice 2 :

1) L'objet fluxEntree représente le traitement du fichier en entrée (fichier source)

Le fluxSortie représente le traitement du fichier en sortie c'est-à-dire de destination

La différence entre les flux de l'exercice précédent est :

Les flux d'E/S étaient définit par des arguments et ces flux utilisés des buffers (temporisation).

On utilise des flux de caractères avant le try. C'est la différence majeure par rapport au programme précédent. On récupère les flux de caractères. Puis on utilise des flux d'octets. Tandis dans le 1er programme on n'utilise pas de flux de caractères.

Les flux de caractères nécessite 2 octets pour coder un caractère d'où le fait qu'on utilise des flux d'octets.

Les flux d'octets permettent l'assemblage de plusieurs flux de caractères sur les fichiers (indispensable).

On a aussi utilisé Scanner et PrintWriter (sont des flux de caractères) au lieu d'InputStream et FileOutputStream (sont des flux d'octets) lors de l'instanciation (*avant le bloc try*)

Remarque :

Scanner permet la lecture d'un flux de caractère tandis que PrintWriter permet l'écriture.

Le programme lit : un fichier source et copie vers un fichier de destination.

La boucle tant que (while) utilise la méthode hasNextLine qui permet de vérifié s'il reste des lignes, et nextLine permet de passer à la ligne suivante. Donc : il copie dans le fichier de destination en numérotant (à partir de 1) par un espace entre le contenu et le numéro. Enfin on ferme les fichiers.

On a aussi utilisé Scanner et PrintWriter. En les chaînant avec l'InputStream et le FileOutputStream dans le bloc try. Cela nous permet de les convertir en chaîne d'octets.

Le bloc catch traite le message d'erreur si le fichier est introuvable.

Il y a évidemment une incrémentation du nombre

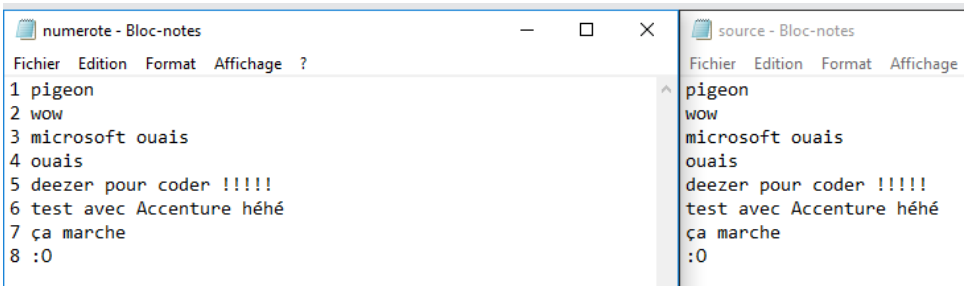
2) Code :

TestES.java

```
1  import java.io.*;
2  import java.util.*;
3
4  public class TestES {
5      public static void main(String[] args) {
6          Scanner fluxEntree = null;
7          PrintWriter fluxSortie = null;
8          try {
9              fluxEntree = new Scanner(new FileInputStream("source.txt"));
10             fluxSortie= new PrintWriter( new FileOutputStream("numerote.txt"));
11         }
12         catch(FileNotFoundException e) {
13             System.out.println("Erreur ouverture fichier.");
14             System.exit(0);
15         }
16         String ligne = null; int no = 0;
17         while (fluxEntree.hasNextLine( )) {
18             ligne = fluxEntree.nextLine( );
19             no++;
20             fluxSortie.println(no + " " + ligne);
21         }
22         fluxEntree.close( );
23         fluxSortie.close( );
24     }
25 }
```

Résultat : On crée un fichier source.txt (au lieu d'original.txt)

Puis on lance le programme le fichier de destination numerote.txt est créé avec le numéro de la ligne



Notes : Si on ne referme pas le fichier d'E/S avec FluxEntree.close() et de même avec FluxSortie alors le fichier apparaît vide.

Résultat en cas d'erreur : En cas d'erreur lors de l'ouverture de fichier ou si le fichier est inexistant :

```
Erreur ouverture fichier.
```

```
12         catch(FileNotFoundException e) {
13             System.out.println("Erreur ouverture fichier."+e);
14             System.exit(0);
15         }
16         String ligne = null; int no = 0;
17         while (fluxEntree.hasNextLine( )) {
<terminated> TestES [Java Application] C:\Program Files\Java\jre-9.0.1\bin\javaw.exe (12 févr. 2019 à 14:45:37)
Erreur ouverture fichier.java.io.FileNotFoundException: source.txt (Le fichier spécifié est introuvable)
```

Avec l'argument e concaténé on identifie l'erreur d'une manière beaucoup plus précise.