11 Manipulation de flux entrée / sortie

Séance 11
Objectif (secondaire) : flux
fin des exos précédents
60, 61, 64

La classe File

Le paquetage `java.io` définit un grand nombre de classes pour gérer les entrés / sorties d'un programme. Parmi elles, la classe `File` permet de manipuler des fichiers ou des répertoires. Une instance de la classe `File` est une représentation logique d'un fichier ou d'un répertoire qui peut ne pas exister physiquement sur le disque. La classe `File` définit notamment les méthodes suivantes :

- `File(String path)`	construit un objet `File` pointant sur l'emplacement passé en paramètre
- `boolean canRead()`	indique si le fichier peut être lu
- `boolean canWrite()`	indique si le fichier peut être modifié
- `boolean createNewFile()`	crée un nouveau fichier vide à l'emplacement pointé par l'objet `File`, `createNewFile()` peut lever l'exception `java.io.IOException`
- `boolean delete()`	détruit le fichier ou le répertoire
- `boolean exists()`	indique si le fichier existe physiquement
- `String getAbsolutePath()`	renvoie le chemin absolu du fichier
- `File getParentFile()`	renvoie un objet `File` pointant sur le chemin parent de celui de l'objet `File` courant
- `boolean isDirectory()`	indique si l'objet `File` pointe sur un répertoire
- `boolean isFile()`	indique si l'objet `File` pointe sur un fichier
- `File[] listFiles()`	si l'objet `File` est un répertoire, renvoie la liste des fichiers qu'il contient
- `boolean mkdir()`	création du répertoire
- `boolean mkdirs()`	création de toute l'arborescence du chemin
- `boolean renameTo(File f)`	renomme le fichier

Exercice 59 – Manipulation de fichiers et d'arborescences

Soit la classe `TestFile` suivante :

```
1 import java.io.File;
2 import java.io.IOException;
3
4 public class TestFile{
5     public static void main(String[] args){
6         try {
7             File f=new File(args[0]);
8             f.delete();
9             System.out.println("Le fichier existe : "+(f.exists()?"oui":"non"));
10            f.createNewFile();
11            System.out.println("Le fichier existe : "+(f.exists()?"oui":"non"));
12            System.out.println(f.getAbsolutePath());
13            System.out.println(f.getPath());
14        } catch(IOException e){
15            System.out.println(e);
16        }
17    }
18 }
```

Q 59.1 Dire ce qu'affiche l'execution suivante : `java TestFile "./2i002/TME11/Files/fichier1.txt"`
- Si le répertoire `"./2i002/TME11/Files"` existe
- Si le répertoire `"./2i002/TME11/Files"` n'existe pas

```
// Si le répertoire existe :
Le fichier existe : non
Le fichier existe : oui
/home/.../2i002/TME11/Files/fichier1.txt
./fichier1.txt

// Si le répertoire n'existe pas
Le fichier existe : non
java.io.IOException: Le chemin d'acces specifie est introuvable

// L'exception est envoyee a la creation du fichier
```

Q 59.2 Modifier la méthode `main` pour qu'il n'y ait plus de problème à la création du fichier

```
// Ajouter les deux lignes suivantes juste apres la construction de f :

File r=f.getParentFile();
if (r!=null){
    r.mkdirs();
}
```

Q 59.3 Écrire une méthode `pwd()` permettant d'afficher le chemin du répertoire courant grâce aux méthodes de la classe `File`

```
public static void pwd(){
        File f=new File(".");
        System.out.println(f.getAbsolutePath());
}
```

Q 59.4 Écrire une méthode `ls(File f)` permettant d'afficher tous les noms de fichiers contenus dans le répertoire passé en paramètre (ne pas afficher les répertoires)

```
public static void ls(File f){
        File[] childs=f.listFiles();
        for(int i=0;i<childs.length;i++){
            File fic=childs[i];
            if (fic.isFile()){
                System.out.println(fic.getAbsolutePath());
            }
        }
}
```

Q 59.5 Écrire une méthode `lsRecursif(File f)` permettant d'afficher tous les noms de fichiers contenus dans l'arborescence prenant sa racine au niveau du répertoire passé en paramètre (ne pas afficher les répertoires)

```
1 public static void lsRecursif(File f){
2         File[] childs=f.listFiles();
3         for(int i=0;i<childs.length;i++){
4             File fic=childs[i];
5             if (fic.isFile()){
6                 System.out.println(fic.getAbsolutePath());
7             } else {
8                 lsRecursif(fic);
9             }
10        }
11 }
```

Les flux

Outre la classe `File`, le paquetage `java.io` (i pour input, o pour output) définit une multitude de classes permettant la manipulation de flux de lecture/écriture. Ces flux permettent des échanges de données entre le programme et d'autres entités, qui peuvent être :

- une variable du programme (par exemple, pour la construction de chaînes de caractères)
- la console de l'utilisateur (`System.in` : entrée standard, `System.out` : sortie standard)
- un fichier (création, lecture, écriture, modifications, ...)
- la mémoire
- ...

Deux categories de flux :

- Les flux entrants pour la lecture
  - `InputStream` pour lire des octets
  - `Reader` pour lire des caractères
- Les flux sortants pour l'écriture
  - `OutputStream` pour écrire des octets
  - `Writer` pour écrire des caractères

Ces classes de flux sont néanmoins des classes abstraites. Les classes à utiliser sont préfixées par :

- la source pour les flux entrants (`FileInputStream`, `FileReader`, `InputStreamReader`, `StringReader`...)
- la destination pour les flux sortants (`FileOutpuStream`, `FileWriter`, `OutputStreamWriter`, `StringWriter`...)

La classe `Reader` définit principalement les méthodes suivantes :

- `void close()`	Ferme le flux
- `int read()`	Lit le caractère suivant du flux et le retourne. Retourne -1 si la fin du fichier est atteinte.
- `int read(char[] cbuf)`	Lit un ensemble de caractères et les place dans le tableau passé en paramètre. Retourne le nombre d'entiers lus, -1 si la fin du fichier est atteinte.
- `long skip(long n)`	Passe un nombre donné de caractères.

La classe `Writer` définit quant à elle les méthodes suivantes :

- `void close()`	Ferme le flux après avoir écrit l'ensemble des caractères en mémoire, `close()` peut lever l'exception `java.io.IOException`
- `void flush()`	Vide la mémoire du flux (force l'écriture de l'ensemble des caractères en mémoire)
- `void write(char c)`	Écrit le caractère c dans le flux.
- `void write(char[] cbuf)`	Écrit l'ensemble des caractère du tableau dans le flux.
- `void write(char[] cbuf, int debut, int nb)`	Écrit nb des caractères du tableau dans le flux en commencant par celui d'index debut.
- `void write(String s)`	Écrit la chaîne de caractère dans le flux.

Il est à noter que l'appel aux méthodes `write()` n'écrit en fait pas les données directement dans la destination pointée par le flux mais passe par une mémoire nommée mémoire tampon. Ce n'est que lorsque celle-ci est pleine ou lors de l'appel à la méthoode `flush()` que l'écriture effective des données est réalisée. Si l'on travaille sur un fichier, l'inscription des données dans ce fichier n'est alors garantie qu'après appel à la méthode `flush()`.

La classe `PrintWriter` simplifie l'utilisation de la classe `Writer` en définissant les méthodes suivantes :

- `PrintWriter(Writer out)`	Construction d'un objet `PrintWriter` sur un flux passé en paramètre
- `void close()`	Ferme le flux
- `void flush()`	Vide la mémoire du flux (force l'écriture de l'ensemble des caractères en mémoire)
- `void print(String s)`	Écrit la chaîne `s` dans le flux. Appel automatique à la methode `flush()`.
- `void println(String s)`	Écrit la chaîne `s` dans le flux avec passage à la ligne. Appel automatique à la methode `flush()`.

Important : pensez à fermer les flux en fin d'utilisation (methode `close()`).

Exercice 60 – Traitement de texte

Rappel : `String` est une classe immutable, c'est-à-dire qu'une variable de type `String` ne peut pas être modifiée. Lorsque l'on pense modifier un objet `String`, en vérité, on crée un nouvel objet `String` à partir de l'ancien.

Q 60.1 Écrire une méthode `String saisie()` qui demande à l'utilisateur de saisir une ligne de texte tant que la ligne entrée par l'utilisateur est différente de la chaîne "_fin_". Cette méthode retourne une chaîne de caractères contenant la concaténation de toutes les lignes saisies. Proposez une première solution utilisant des concaténations de `String`. Puis proposez une deuxième solution utilisant un seul objet `StringWriter`.

Il existe 4 façons de concaténer des chaines de caractères : l'opérateur "+", la méthode `String.concat()`, et les classes `StringBuffer` et `StringBuilder`. Mais avant de les comparer, il faut comprendre la base du problème : **l'immutabilité** de la classe `String`. `String` étant une classe immutable, une variable de type `String` ne peut pas être modifiée. Lorsque l'on croit modifier une `String`, en vérité, on crée une nouvelle `String` à partir de l'ancienne qui sera soumise au garbage collector.
Exemple : `String a="Bonjour"; a=a+" tous le monde";` est équivalent à :
`String a=new String("Bonjour");a=new String(a+" tous le monde");`
Le premier objet est détruit lorsqu'on crée le deuxième.

```
1 public static String saisie(){
2         String texte="";
3         String ligne="";
4         while((ligne=Clavier.saisirLigne("Entrez Texte :")).compareTo("_fin_")!=0){
5             texte+=ligne+"\n";
6         }
7         return(texte);
8 }
9
10 // On peut faire remarquer qu'il serait plus efficace d'utiliser un StringWriter
11 public static String saisie(){
```

```
12          StringWriter texte=new StringWriter();
13          String ligne="";
14          while((ligne=Clavier.saisirLigne("Entrez Texte :")).compareTo("_fin_")!=0){
15              texte.write(ligne+"\n");
16          }
17          return(texte.toString());
18  }
19
20  // ou un java.lang.StringBuilder
21  public static String saisie(){
22          StringBuilder texte=new StringBuilder();
23          String ligne="";
24          while((ligne=Clavier.saisirLigne("Entrez Texte :")).compareTo("_fin_")!=0){
25              texte.append(ligne+"\n");
26          }
27          return(texte.toString());
28      }
```

Q 60.2 Écrire une méthode `affiche(String fichier)` affichant le contenu du fichier dont le nom est passé en paramètre.

```
1  public static void affiche(String fichier) throws IOException{
2          File file=new File(fichier);
3          FileReader lecteur=new FileReader(file);
4          try {
5              int c;
6              StringBuilder texte=new StringBuilder();
7              while((c=lecteur.read())!=-1){
8                  texte.append((char)c);
9              }
10             System.out.println(texte.toString());
11         }
12         finally{
13             lecteur.close();
14         }
15 }
16
17 // --- Ou avec un tableau de char ---
18 public static void affiche(String fichier) throws IOException{
19         File file=new File(fichier);
20         FileReader lecteur=new FileReader(file);
21         try {
22             int nb=0;
23             char[] buf=new char[100];
24             StringBuilder texte=new StringBuilder();
25             while((nb=lecteur.read(buf))>0){
26                 texte.append(buf);
27                 buf=new char[100];
28             }
29             System.out.println(texte.toString());
30         }
31         finally{
32             lecteur.close();
33         }
34 }
```

Q 60.3 Écrire une méthode `afficheLignes(String fichier)` affichant, en numérotant les lignes, le contenu du fichier passé en paramètre.

```
public static void afficheLignes(String fichier) throws IOException{
        File file=new File(fichier);
        FileReader lecteur=new FileReader(file);
        try {
            int c;
            int nbLines=1;
            StringBuilder texte=new StringBuilder();
            boolean nligne=true;
            while((c=lecteur.read())!=-1){
                if (nligne){
                    texte.append(nbLines+" : ");
                    nbLines++;
                    nligne=false;
                }
                char car=(char) c;
                texte.append(car);
                if (car=='\n'){
                    nligne=true;
                }
            }
            System.out.println(texte.toString());
        }
        finally{
            lecteur.close();
        }
}
```

Q 60.4 Écrire une méthode `ecrireTexte(String fichier)` permettant de créer un nouveau fichier contenant un texte saisi par l'utilisateur.

```
public static void ecrireTexte(String fichier) throws IOException{

        File file=new File(fichier);
        FileWriter ecrivain=new FileWriter(file);
        try {
            String saisie=saisie();
            ecrivain.write(saisie);
            ecrivain.flush(); // ecriture dans le fichier maintenant
        }
        finally{
            ecrivain.close();
        }
}
```

Q 60.5 Écrire une méthode `ajouteTexte(String fichier)` permettant d'ajouter, en fin de fichier passé en paramètre, du texte saisi par l'utilisateur.

```
public static void ajouteTexte(String fichier) throws IOException{

```

```
3         File file=new File(fichier);
4         File tmp=new File("tmp_file.txt");
5
6         FileWriter ecrivain=new FileWriter(tmp);
7         FileReader lecteur=new FileReader(file);
8         try {
9                 int nb=0;
10                char[] buf=new char[100];
11                while((nb=lecteur.read(buf))>0){
12                    ecrivain.write(buf,0,nb-1);
13                    ecrivain.write("\n");
14                    ecrivain.flush();
15                    buf=new char[100];
16
17                }
18
19                String saisie=saisie();
20                ecrivain.write(saisie);
21                ecrivain.flush();
22        }
23        finally{
24                ecrivain.close();
25                lecteur.close();
26                if (!file.delete()){
27                    throw new IOException("Probleme suppression fichier");
28                }
29                if (!tmp.renameTo(file)){
30                    throw new IOException("Probleme renommage du fichier");
31                }
32        }
33}
34
35// Ou avec le constructeur de FileWriter qui accepte en second argument un boolean
      append
36
37public static void ajouteTexte(String fichier) throws IOException{
38
39        File file=new File(fichier);
40        FileWriter ecrivain=new FileWriter(file,true);
41
42        try {
43                String saisie=saisie();
44                ecrivain.write(saisie);
45                ecrivain.flush();
46        }
47        finally{
48                ecrivain.close();
49        }
50}
```

Q 60.6 Écrire une méthode `replace(int num, String newLigne, String fichier)` permettant de remplacer, dans le fichier passé en paramètre, la ligne numéro `num` par la nouvelle ligne `newLigne`.

```
1public static void replace(int num, String newLigne, String fichier) throws IOException
      {
2         File file=new File(fichier);
```

```
3         File tmp=new File("tmp_file.txt");
4
5         if (file.exists()){
6             FileWriter ecrivain=new FileWriter(tmp);
7             FileReader lecteur=new FileReader(file);
8             try {
9                 int c;
10                int nbLines=0;
11                StringBuilder ligne=new StringBuilder();
12
13                while((c=lecteur.read())!=-1){
14                    char car=(char) c;
15                    ligne.append(car);
16                    if (car=='\n'){
17                        nbLines++;
18                        if (nbLines==num){
19                            ecrivain.write(newLigne+"\n");
20                        } else {
21                            ecrivain.write(ligne.toString());
22                        }
23                        ligne=new StringBuilder();
24                    }
25                }
26                if (nbLines<num) {
27                    System.out.println("Pas de ligne "+num);
28                }
29            }
30            finally{
31                ecrivain.close();
32                lecteur.close();
33                if (!file.delete()){
34                    throw new IOException("Probleme suppression fichier");
35                }
36                if (!tmp.renameTo(file)){
37                    throw new IOException("Probleme renommage du fichier");
38                }
39            }
40        } else {
41            System.out.println("Pas de ligne "+num);
42        }
43
44    }
```

Q 60.7 Écrire un programme proposant à l'utilisateur un menu lui permettant d'éditer un fichier dont le chemin est passé en argument. Exemple :

```
Fichier "Texte.txt"
1. Ajouter texte
2. Afficher fichier
3. Remplacer ligne
4. Quitter
```

```
1 public static int menu(String fichier){
2         System.out.println("Fichier "+fichier);
3         System.out.println("1. Ajouter texte");
4         System.out.println("2. Afficher fichier");
```

```
5         System.out.println("3. Remplacer ligne");
6         System.out.println("4. Quitter");
7         int choix=Clavier.saisirEntier("Votre choix ?");
8         return(choix);
9 }
10
11 public static void main(String[] args){
12         int choix;
13         String fichier=args[0];
14         try {
15             File file=new File(fichier);
16             if (!file.exists()){
17                 File parent=file.getParentFile();
18                 if (parent!=null){
19                     parent.mkdirs();
20                 }
21                 file.createNewFile();
22             }
23
24             while((choix=menu(fichier))!=4){
25                 if ((choix<1) || (choix>4)){
26                     System.out.println("Choix invalide");
27                     continue;
28                 } else if (choix==1){
29                     ajouteTexte(fichier);
30                 } else if (choix==2){
31                     afficheLignes(fichier);
32                 } else if (choix==3){
33                     int num=Clavier.saisirEntier("Quele ligne a remplacer ?");
34                     String newLigne=Clavier.saisirLigne("Entrez la nouvelle ligne "+num
                           );
35                     replace(num,newLigne,fichier);
36                 }
37             }
38         } catch(IOException e) {
39             System.out.println(e);
40         }
41 }
```

Exercice 61 – Copie de fichiers binaires

Q 61.1 Écrire un programme permettant de copier un fichier binaire passé en premier argument sous le nom passé en second.

```
1 import java.io.*;
2
3 public class CopyFiles {
4    public static void copyFile(File srcFile, File destFile) throws IOException {
5       InputStream oInStream = new FileInputStream(srcFile);
6       OutputStream oOutStream = new FileOutputStream(destFile);
7
8       // Transfer bytes from in to out
9       byte[] oBytes = new byte[1024];
10      int nLength;
```

```
11
12        while ((nLength = oInStream.read(oBytes)) > 0) {
13           oOutStream.write(oBytes, 0, nLength);
14           oOutStream.flush();
15        }
16        oInStream.close();
17        oOutStream.close();
18     }
19     public static void main(String[] args){
20        try {
21           copyFile(new File(args[0]),new File(args[1]));
22        } catch(IOException e) {
23           System.out.println(e);
24        }
25     }
26 }
```

La mise en mémoire tampon

La mise en mémoire tampon des données lues permet d'améliorer les performances des flux sur une entitée. Par l'utilisation directe d'un objet `Reader`, les caractères sont lus un par un dans le flux, ce qui est très peu efficace. La classe `BufferedReader` (existe aussi pour `BufferedInputStream` pour les octets) permet la mise en mémoire tampon des données lues avant transmission au programme.
En outre, elle simplifie l'utilisation du `Reader` en définissant notamment une méthode `String readLine()` permettant de lire les données ligne après ligne plutôt que caractère après caractère (toutes les méthodes de `Reader` sont disponibles dans cette classe mais avec une meilleure gestion de la mémoire).

Exercice 62 – Mise en mémoire tampon

Q 62.1 Sachant que la construction d'un `BufferedReader` se fait en passant un flux `Reader` en paramètre, écrivez l'ouverture d'un flux de lecture avec utilisation de la mémoire tampon sur un fichier "text.txt" du répertoire courant.

```
1 BufferedReader in= new BufferedReader(new FileReader(new File("text.txt")));
```

Q 62.2 Écrire une méthode `afficheLignesFichier(String fichier)` qui affiche ligne après ligne le texte du fichier dont le chemin est passé en paramètre.

```
1 public static void afficheLignesFichier(String fichier) throws IOException,
     FileNotFoundException{
2         BufferedReader lecteur=new BufferedReader(new FileReader(new File(fichier)));
3         try {
4             String ligne="";
5             while((ligne=lecteur.readLine())!=null){
6                 System.out.println(ligne);
7             }
8         }
9         finally{
10                lecteur.close();
11        }
12 }
```

Q 62.3 Sachant qu'il est également recommandé par soucis d'efficacité d'encapsuler tout flux en écriture dans un objet `BufferedWriter` (resp. `BufferedStream` pour l'écriture d'octets), écrire une classe `Ecrivain` ouvrant un flux en écriture sur un fichier à sa construction et disposant des méthodes données ci-dessus pour la classe `PrintWriter` (sauf méthode `flush()`). On pourra donner une version avec héritage et une version sans.

```java
// --- Avec heritage ---
import java.io.*;
public class Ecrivain extends PrintWriter {
    public Ecrivain(String fichier) throws IOException{
        super(new BufferedWriter(new FileWriter(new File(fichier))));
    }
    public void finalize(){
        close();
    }
}

// --- Sans heritage ---
import java.io.*;
public class Ecrivain {
    private PrintWriter pw=null;
    public Ecrivain(String fichier) throws IOException{
        pw=new PrintWriter(new BufferedWriter(new FileWriter(new File(fichier))));
    }
    public void close(){
        if (pw!=null){
            pw.close();
            pw=null;
        }
    }
    public void finalize(){
        close();
    }
    public void println(String s){
        if (pw!=null){
            pw.println(s);
        } else {
            System.out.println("Flux ferme");
        }
    }
    public void print(String s){
        if (pw!=null){
            pw.print(s);
        } else {
            System.out.println("Flux ferme");
        }
    }
}
```

Exercice 63 – Production automatique de compte rendu TME

L'objectif de cet exercice est d'utiliser les connaissances acquises sur la lecture et l'écriture de fichier pour programmer un outil de production automatique de compte rendu de TME.
On considère que l'utilisateur dispose d'une arborescence (telle que vous devez l'avoir) prenant racine en un répertoire 2i002. Ce répertoire contient un répertoire par TME (numérotés de TME1 à TME11), chacun d'entre eux contenant eux mêmes un répertoire par exercice (Exo1, Exo2, ... ExoN). On considère également que l'on dispose d'un fichier "etudiants.txt" dans le répertoire 2i002 contenant les prenoms, noms et numeros d'etudiants des utilisateurs du

programme (une ligne par étudiant). Le fichier doit se terminer par une ligne "Groupe : <numero du groupe>". Enfin, chaque répertoire d'exercice contient deux fichiers "intitule.txt" et "executions.txt", le premier contenant l'énoncé de l'exercice, le second contenant les résultats d'exécution des programmes ainsi que les observations qui ont pu avoir été faites.

Q 63.1 Écrire un programme `RenduTMEProducer` prenant en argument le chemin du répertoire de TME concerné par le compte rendu et produisant en racine de ce répertoire un fichier "compteRenduTME.txt" de la forme de celui que vous avez l'habitude de rendre en fin de TME.

```
1 import java.io.*;
2 import java.util.ArrayList;
3 public class RenduTMEProducer {
4
5     public static ArrayList<File> getAllJavaFiles(File f){
6         ArrayList<File> ret=new ArrayList<File>();
7         File[] childs=f.listFiles();
8         for(int i=0;i<childs.length;i++){
9             File fic=childs[i];
10            if (fic.isFile()){
11                String path=fic.getAbsolutePath();
12                int iext=path.lastIndexOf(".");
13
14                if ((iext>0) && (iext<path.length()-4)){
15                    String ext=path.substring(iext+1,iext+5);
16                    if (ext.compareTo("java")==0){
17                        ret.add(fic);
18                    }
19                }
20            } else {
21                ret.addAll(getAllJavaFiles(fic));
22            }
23        }
24        return(ret);
25    }
26
27    public static String getTexteFromFile(File f) throws IOException{
28        StringBuilder sb=new StringBuilder();
29        BufferedReader lecteur=new BufferedReader(new FileReader(f));
30        try {
31            String ligne="";
32            while((ligne=lecteur.readLine())!=null){
33                    sb.append(ligne+"\n");
34            }
35        }
36        finally{
37            lecteur.close();
38        }
39        return(sb.toString());
40    }
41    public static void produceCompteRendu(String rep) throws IOException{
42        File tme=new File(rep);
43        if (!tme.exists()){
44            throw new IOException("Le repertoire de TME donne n existe pas");
45        }
46        tme=new File(tme.getAbsolutePath());
47        if (!tme.isDirectory()){
48            throw new IOException("Le chemin donne n est pas un repertoire");
49        }
50
```

```
51          File 2i002=tme.getParentFile();
52          if (2i002.getName().compareTo("2i002")!=0){
53              throw new IOException("Le repertoire parent du repertoire donne n'est pas 2
                    i002");
54          }
55
56          File etudiants=new File(2i002.getAbsolutePath()+"/etudiants.txt");
57          if (!etudiants.exists()){
58              throw new IOException("Le repertoire 2i002 ne contient pas de fichier nomme
                    etudiants.txt");
59          }
60
61          File rendu=new File(tme.getAbsolutePath()+"/compteRenduTME.txt");
62          System.out.println(rendu.getAbsolutePath());
63          boolean ok=true;
64          if (rendu.exists()){
65              ok=false;
66              String ecrase=Clavier.saisirLigne("Le repertoire "+rep+" contient deja un
                    compte rendu de TME. Voulez vous l'ecraser ?(Oui/Non)");
67              ecrase=ecrase.toLowerCase();
68              if (ecrase.compareTo("oui")==0){
69                  ok=true;
70              }
71          }
72          if (ok){
73              PrintWriter ecrivain=new PrintWriter(new BufferedWriter(new FileWriter(
                    rendu)));
74              try {
75                  ecrivain.println("Compte Rendu 2i002 \n");
76                  String texteEtudiants=getTexteFromFile(etudiants);
77                  ecrivain.println(texteEtudiants+"\n");
78                  ecrivain.println("Numero du TME : "+tme.getName()+"\n");
79
80                  File[] exos=tme.listFiles();
81                  for(int i=0;i<exos.length;i++){
82                      File exo=exos[i];
83                      if (exo.isDirectory()){
84                          System.out.println("exo : "+exo.getAbsolutePath());
85                          ecrivain.println("// ----------------------");
86                          ecrivain.println("Intitule de l'exercice :");
87                          File intitule=new File(exo.getAbsolutePath()+"/intitule.txt");
88                          if (intitule.exists()){
89                              ecrivain.println(getTexteFromFile(intitule));
90                          }
91                          ecrivain.println("");
92                          ecrivain.println("Classes : \n");
93                          ArrayList<File> classes=getAllJavaFiles(exo);
94                          for(File classe:classes){
95                              ecrivain.println(getTexteFromFile(classe)+"\n");
96                          }
97                          ecrivain.println("Executions et observations :");
98                          File execution=new File(exo.getAbsolutePath()+"/executions.txt"
                                );
99                          if (execution.exists()){
100                             ecrivain.println(getTexteFromFile(execution));
101                         }
102                         ecrivain.println("");
103                     }
104                 }
```

```
105             } catch(IOException e) {
106                 ecrivain.close();
107                 rendu.delete();
108                 throw(e);
109             }
110             finally{
111                 ecrivain.close();
112             }
113         }
114     }
115
116     public static void main(String[] args){
117         try {
118             produceCompteRendu("ExempleArborescence/2i002/TME11");
119         } catch(IOException e) {
120             System.out.println(e+"\n");
121             e.printStackTrace();
122         }
123
124     }
125 }
```

Entrée / Sortie standard

Nous avons vu la manière d'ecrire ou lire dans des fichiers. L'écriture sur la sortie standard (tel qu'on l'a souvent pratiqué par `System.out.println` sans trop savoir à quoi cela correspondait) ou la lecture à partir de l'entrée standard (comme ce que l'on fait avec la classe `Clavier` pour interagir avec l'utilisateur) utilisent également des flux en lecture/écriture :

- La sortie standard `System.out` correspond à un flux `PrintWriter` (c'est pourquoi on peut utiliser la méthode `println` sur cet objet)
- L'entrée standard `System.in` correspond à un flux `InputStream` (flux permettant de lire des octets à partir d'une source)

Pour la sortie, aucun problème, on sait déja le faire : `System.out.println("texte a afficher");`
Pour l'entrée, c'est un peu plus compliqué : il s'agit de transformer les octets lus à partir de l'objet `InputStreamReader` en caractères que l'on sait manipuler.

Exercice 64 – Classe Clavier

Q 64.1 Sachant que le paquetage `java.io` contient une classe de flux `InputStreamReader` permettant de lire des caractères à partir d'un flux entrant d'octets, réécrire le code de la classe `Clavier`, notamment :

- La fonction statique `String SaisirLigne(String message)`
- La fonction statique `int SaisirEntier(String message)`

```
1 import java.io.* ;
2
3 /**Cette classe implante des saisies au clavier par lecture d'une ligne. */
4 public class Clavier {
5
6     private static final BufferedReader in =
7         new BufferedReader (new InputStreamReader (System.in)) ;
8
9     /** Affiche le message et retourne un int lu au clavier. */
```

```
10      public static int saisirEntier (String mess){
11          while(true){
12              try {
13                  return Integer.parseInt (saisirLigne ( mess)) ;
14              } catch (NumberFormatException e) {
15                  mess = "Recommencez␣:␣" ;
16              }
17          }
18      }
19
20      /** Affiche le message et retourne une ligne lue au clavier. */
21      public static String saisirLigne (String mess) {
22          System.out.println (mess) ;
23          try {
24              return in.readLine () ;
25          } catch (IOException e){
26              return null; // provisoire !!
27          }
28      }
29  } // Clavier
```

Quizz 14 – `String`, classe immutable

QZ 14.1 Combien d'objets sont créés dans les instructions ci-après ?
`String a="Bonjour"; a=a+" tous le monde";`

Explications à vérifier.
Il y a deux objets créés. `String a="Bonjour"; a=a+" tous le monde";` est équivalent à :
`String a=new String("Bonjour");a=new String(a+" tous le monde");`
Le premier objet est détruit lorsqu'on crée le deuxième.

QZ 14.2 Donnez une solution équivalente en utilisant un `StringWriter`.

Un seul objet nécessaire.

```
1 import java.io.StringWriter;
2
3 StringWriter sw=new StringWriter();
4 sw.write("Bonjour");
5 sw.write("␣tous␣le␣monde");
6 System.out.println(sw.toString());
```

Aide mémoire

Convention d'écriture

- Le nom des classes (et des constructeurs) commence par une majuscule ;
- Le nom des méthodes, des variables ou des instances commence par une minuscule ;
- Les mots réservés sont obligatoirement en minuscules ;
- Les constantes sont généralement en majuscules.