R2.01 : cours n°10 I.Borne

- Entrées/sorties standards
- Manipulation des chaînes de caractères avec StringBuilder
- Archive jar

La gestion des fichiers

- La gestion de fichiers proprement dire se fait par l'intermédiaire de la classe `File` (java.io.File)
- Elle possède des méthodes pour interroger ou agir sur le système de gestion de fichiers du système d'exploitation.
- Un objet de la classe `File` peut représenter un fichier ou un répertoire.

Exemple : lister les fichiers d'un répertoire

```
import java.io.File;
public class Listeur {
  public static void main(String[] args) {
     File rep = new File(".");    // répertoire courant
     if (rep.isDirectory()) {
          String t[]=rep.list(); //liste les fichiers du répertoire
          for (String e : t)
               System.out.println(e);  // imprime la liste de fichiers
     }
  }
}
```


Flux d'entrée/sortie

- En java les classes pour les E/S sont dans le package **`java.io`**.
- Les deux classes principales sont :
  - **`InputStream`** : classe abstraite, définit les fonctions de lecture (entrée ou input en anglais),
  - **`OutputStream`** : classe abstraite, définit les fonctions d'écriture (sortie ou output en anglais).
- Pour les fichiers :
  - **`FileInputStream`** : lecture d'un fichier, hérite de **`InputStream`**
  - **`FileOutputStream`** : écriture d'un fichier, hérite de **`OutPutStream`**
- La classe **`java.io.PrintStream`** hérite de la classe **`java.io.OutputStream`**, et permet d'afficher tous les types de données sous forme textuelle.
- La sortie standard et l'erreur standard impriment sur la console et sont des instances de la classe **`java.io.PrintStream`**.

Les classes de gestion des flux

- Il faut comprendre la dénomination de ces classe pour bien les utiliser. Le nom se compose d'un préfixe et d'un suffixe :
  - 4 suffixes en fonction du type de flux (octets ou caractères)
  - Sens du flux (entrée ou sortie)

	Flux d'octets	Flux de caractères
Flux d'entrée	InputStream	Reader
Flux de sortie	OutputStream	Writer

les sous-classes de Reader sont des types de flux en lecture sur des ensembles de caractères
les sous-classes de Writer sont des types de flux en écriture sur des ensembles de caractères
les sous-classes de InputStream sont des types de flux en lecture sur des ensembles d'octets
les sous-classes de OutputStream sont des types de flux en écriture sur des ensembles d'octets

Les flux de caractères (package java.io)

- Ils sont représentés par les sous-classes de **`Reader`** et **`Writer.`**
- Ces flux utilisent le codage de caractères Unicode
- Pour écrire des chaînes de caractères et des nombres sous forme de texte :
  - La classe **`PrintWriter`** possède un certain nombre de méthodes print(...), println...)
- Pour lire des chaînes de caractères sous forme de texte :
  - Soit **`BufferedReader`** qui possède une méthode readLine()
  - Ou plus simple la classe **`Scanner`** qui peut se brancher sur à peu près n'importe quelle source et bien sûr sur une simple String

Retour sur le lecture au clavier

- Problème du « bon » type de la donnée lue.
- `InputMismatchException` si le type lu n'est pas celui attendu.
- La classe Scanner possède un ensemble de méthodes pour vérifier le prochain token avant de le lire :
  - `hasNextInt() , hasNextDouble(), hasNextBoolean()`...
- Pour ensuite réaliser la lecture avec :
  - `int nextInt(), double nextDouble(), boolean nextBoolean()`
- Penser également à la méthode `useDelimiter()`à la création du scanner.

Exemple de lecture au clavier

```
import java.util.Scanner;

public class TypeSafeInteger {

  public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    String garbage;
    System.out.print("Enter your age as an integer >");
    while (! scan.hasNextInt()) { // il faut taper un nombre
      garbage = scan.nextLine();
      System.out.print("\nPlease enter an integer >");
    }
    int age = scan.nextInt();
  }
}
```

La lecture des fichiers texte (flux de caractères)

- Des fichiers créés avec un éditeur de texte simple, ou du code source Java ou des fichiers HTML.
- Pour lire un fichier de texte, il faut définir un objet de type **`FileReader`** avec le nom du fichier à lire :

```
FileReader  reader = new FileReader("input.txt");
```

(lance FileNotFoundException si le fichier n'existe pas ou est un répertoire)

- Ensuite on le relie à un objet Scanner pour la lecture.

```
Scanner scan = new Scanner(reader);
```

- La lecture se fait ainsi en envoyant des messages à l'objet scanner, comme on le fait habituellement.
- Penser au chemin pour le nom de fichier, par exemple : `"../data/input.txt"`

Lecture de fichier avec `java.util.Scanner`

- Une fois le flux d'entrée connecté à un objet Scanner, c'est à lui que sont envoyés les messages de lecture.
- Si le flux de lecture est structuré : AA123, 230,45
- Avec l'exemple précédent, on indique que les données sont séparées par des virgules :

```
scan.useDelimiter(",") ;
```

- Les principales méthodes pour lire un fichier :
  - **`public String nextLine():`** retourne le reste de la ligne courante excepté le séparateur de ligne et se positionne au début de la prochaine ligne.
  - **`public boolean hasNextLine() :`** retourne vrai s'il y a une autre ligne en entrée, faux sinon.
  - **`public void close()`** : s'il n'est pas déjà fermé, invoque la méthode close de la source de lecture (le fichier).

Exemple : lecture de fichier

```
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class Lecteur {

  public static void main(String[] args) {
    String fileName = "listeur.java"; //
    try {
      //  ouverture du fichier
      Scanner in = new Scanner (new FileReader (fileName)) ;
      //  lecture et impression des lignes une par une
      while (in.hasNextLine() ) {
        System.out.println(in.nextLine ()) ;
      }
      // fermeture du fichier ouvert en lecture
      in.close() ;
   } catch (FileNotFoundException e) {
       System.out.println ("readFile - Fichier non trouve : "
                           + fileName) ; }
 }
}
```


Ecrire du texte dans un fichier

- La classe **`PrintWriter`** facilite l'écriture de texte dans des flux de sortie.
- Elle implémente toutes les méthodes `print` de **`PrintStream`**
- Les méthodes de cette classe ne lancent pas d'exception d'entrée/sortie.
- Mais le constructeur lance ***FileNotFoundException*** sil n'arrive pas à écrire dans un fichier existant ou ne peut pas le créer.
- Comme pour la lecture il faut fournir un nom de fichier au constructeur.

Ecriture avec `java.io.PrintWriter`

- Imprime des représentations formatées des objets
- Implémente toutes les méthodes `print` de `PrintStream`
- Les méthodes pour les fichiers :

```
public PrintWriter(String fileName)
                throws FileNotFoundException
```

Crée un nouveau **`PrintWriter`** avec le nom de fichier spécifié et crée l'**`OutputStreamWriter`** intermédiaire nécessaire.

`public void close()` : ferme le flux et relâche les ressources systèmes associées.

Ecriture dans un nouveau fichier avec **`PrintWriter`**

- Pour écrire un fichier de texte, il faut définir un objet de type `PrintWriter` avec le nom du fichier à écrire :

```
PrintWriter  out = new PrintWriter("output.txt");
```

- Si le fichier existe déjà, il est vidé avant que les nouvelles données soient écrites.
- Si le fichier n'existe pas, un fichier vide est créé.
- Il suffit d'utiliser les méthodes `print` et `println` pour envoyer des nombres, des objets et des chaînes de caractères à un `PrintWriter` :

```
out.println(29.95);
out.println(New Rectangle(5,10,15,25));
out.println("Bonjour !");
```


Ecriture d'une collection dans un fichier (v1)

```
public static void writeFile(ArrayList<String> liste, String
fileName) {
    try {
      // ouverture du fichier
      PrintWriter out = new PrintWriter (fileName) ;
      // ecriture dans le fichier
      for (String ligne : liste)
        out.println (ligne) ;
      // fermeture du fichier
      out.close() ;

    } catch (FileNotFoundException e) {
          System.out.println (e.getMessage()) ;

    }
}
```


Ecriture d'une collection dans un fichier (v2)

```
public static void writeFile(ArrayList<String> liste, String
fileName) throws FileNotfoundException {

      // ouverture du fichier
      PrintWriter out = new PrintWriter (fileName) ;
      // ecriture dans le fichier
      for (String ligne : liste)
        out.println (ligne) ;
      // fermeture du fichier
      out.close() ;
}
```

Ajouter des caractères à un fichier

- Si on veut ré-ouvrir un fichier pour ajouter encore du texte il faut utiliser un *FileWriter* couplé à un *PrintWriter*.

```
public FileWriter(String fileName, boolean append)
           throws IOException
```

`append` - boolean if `true`, then data will be written to the end of the file rather than the beginning

IOException - if the named file exists but is a directory rather than a regular file, does not exist but cannot be created, or cannot be opened for any other reason.

Exemple :

```
FileWriter fw = new FileWriter(fileName, true);
PrintWriter out = new PrintWriter (fw);
```


La fermeture d'un fichier

- Il faut fermer tous les fichiers que l'on a utilisés en envoyant le message close() :

```
in.close() ;
out.close();
```

Il faut fermer aussi les instances de Scanner.

Les exceptions pour les E/S de fichiers

- Nous devons gérer les exceptions qui peuvent être lancées quand on utilise des fichiers.
- Quand le fichier en entrée ou en sortie n'existe pas, les constructeurs de **`FileReader`** et de **`PrintWriter`** lancent une exception du type `FileNotFoundException`.
- La méthode `close` du `FileReader`, ou les constructeurs de `FileWriter` peuvent lancer une exception du type `IOException` .

Les imports de classes

- Les classes concernant les fichiers se trouvent dans le package java.io, il faut penser à importer ceux qui vont nous servir :

```
import java.io.FileReader;
import java.io.Filewriter;
import java.io.PrintWriter;
```

- Les classes d'exception pour les fichiers sont à importer :

```
import java.io.FileNotFoundException;
import java.io.IOException;
```

- Si on veut utiliser un Scanner il faut ajouter :

```
import java.util.Scanner;
```

Manipulation des chaînes de caractères avec `StringBuilder`

La classe `StringBuilder`

- Les objets `StringBuilder` sont comme les `String` mais ils peuvent être modifiés.
- Cette classe s'utilise quand on a besoin de concaténer un certain nombre de chaînes de caractères.
- Chaque instance contient une capacité qui est le nombre de caractères alloués.
- Le constructeur
  - **`public StringBuilder()`** : crée un objet vide avec une capacité de 16
  - **`public StringBuilder(String str)`** : crée un objet initialisé avec le contenu de la chaîne spécifiée en paramètre

Les principales méthodes

- Ajout de caractères
  - **`public StringBuilder append(char c)`**
  - **`public StringBuilder append(String str)`**
  - **`public StringBuilder insert(int offset,char c)`**
- Modification, recherche
  - **`public void setCharAt(int index,char ch)`**
  - **`public StringBuilder reverse()`**
  - **`public char charAt(int index)`**
  - **`public int indexOf(String str)`**
  - **`public int lastIndexOf(String str)`**
  - **`public int length()`**
  - **`public String toString()`**

Exemple

- Documentation sur :

https://docs.oracle.com/javase/tutorial/java/data/buffers.html

Exemple : palindrome

```
public class StringBuilderDemo {
    public static void main(String[] args) {
        String palindrome = "esope reste ici et se repose";
        StringBuilder sb = new StringBuilder(palindrome);
        sb.reverse();  // reverse it
        System.out.println(sb);
    }
}
```

Création d'archive Java

Archives jar

- En informatique, un fichier JAR (Java ARchive) est un fichier ZIP utilisé pour distribuer un ensemble de classes Java.
- Ce format est utilisé pour stocker les définitions des classes, ainsi que des métadonnées, constituant l'ensemble d'un programme.
- Les fichiers JAR sont créés et extraits à l'aide de la commande jar incluse dans le JDK.
- On peut renommer les fichiers .jar avec l'extension.zip et les manipuler avec les outils ZIP. La classe Java JarFile du package java.util.jar hérite de ZipFile.

Les opérations usuelles pour les fichiers jar

- Les commandes sont similaires à celles du tar Unix :
  - Créer un fichier JAR
    - `jar cf jar-file input-files(s)`
  - Voir le contenu
    - `jar tf jar-file`
  - Extraire le contenu
    - `jar xf jar-file`
  - Exécuter une application contenue dans l'archive (il faut un entête Main-class dans le manifest)
    - `java -jar app.jar`

Pour rendre un jar exécutable

- Il faut indiquer le point d'entrée de l'application qui sera une classe ayant une méthode *main*, et fournir cette information au fichier manifest.
- Exemple : pour exécuter la méthode main de la classe **MyClass** qui est dans le package **myPackage** (contenant les .class de l'application) :
- Il faut créer un fichier texte nommé Manifest.txt et dont le contenu est :

```
Main-Class: myPackage.MyClass
```

Attention le fichier doit se terminer avec un retour à la ligne (ligne vide)

- Création du fichier JAR nommé MyJar.jar :

```
jar cfm MyJar.jar Manifest.txt myPackage/*.class
```

- Pour l'exécuter :

```
java -jar MyJar.jar
```

Exécution d'une archive jar

- Pour exécuter un tel fichier JAR, il faut entrer la ligne de commande suivante :

  java -jar <nom_du_fichier_jar>.jar

- Pour en savoir plus sur jar :

  http://docs.oracle.com/javase/tutorial/deployment/jar/