Programmation `Java` : cours 3

Valeur d'accueil et de reconversion en informatique (VARI1)
Daniel Porumbel (dp.cnam@gmail.com)
`http://cedric.cnam.fr/~porumbed/vari1/`

Exemple d'écriture de fichier

```
public static void main(String[] args){
  try{          //n'oublier pas import java.io.*
    BufferedWriter sortie
         = new BufferedWriter(
                new FileWriter("sortie.txt"));
    sortie.write("ligne un");
    sortie.newLine();              //sauter ligne
    sortie.write("ligne deux");
    sortie.newLine();
    sortie.write("ligne trois");
    sortie.close();
  }catch(Exception e){
    System.out.println("Exception: "+e);
  }
}
```

Exemple d'écriture de fichier

```
public static void main(String[] args){
  try{          //n'oublier pas import java.io.*
    BufferedWriter sortie
         = new BufferedWriter(
                 new FileWriter("sortie.txt"));
    sortie.write("ligne un");
    sortie.newLine();              //sauter ligne
    sortie.write("ligne deux");
    sortie.newLine();
    sortie.write("ligne trois");
    sortie.close();
  }catch(Exception e){
    System.out.println("Exception: "+e);
  }
}
```

Rappel : le bloc try-catch est obligatoire pour d'éventuelles exceptions (impossible d'accéder au fichier)

Copier un fichier

```
try{                        //n'oublier pas import java.io.*
    BufferedReader lecteur
        = new BufferedReader(new FileReader("a.txt"));
    BufferedWriter sortie
        = new BufferedWriter(new FileWriter("b.txt"));

    String ligne = lecteur.readLine();//readLine()
    while(ligne!=null){                 //renvoie null
        sortie.write(ligne);            //null à la fin
        sortie.newLine();               //du fichier
        ligne = lecteur.readLine();
    }

    sortie.close();
    lecteur.close();
}catch(Exception e){
    System.out.println("Exception: "+e);
}
```

Copier un fichier

```
try{                          //n'oublier pas import java.io.*
    BufferedReader lecteur
        = new BufferedReader(new FileReader("a.txt"));
    BufferedWriter sortie
        = new BufferedWriter(new FileWriter("b.txt"));
```

Copier ligne par ligne

```
    String ligne = lecteur.readLine();//readLine()
    while(ligne!=null){               //renvoie null
        sortie.write(ligne);          //null à la fin
        sortie.newLine();             //du fichier
        ligne = lecteur.readLine();
    }

    sortie.close();
    lecteur.close();
}catch(Exception e){
    System.out.println("Exception: "+e);
}
```

Faire l'utilisateur saisir le fichier à copier

```
//n'oublier au début import java.util.*;
public static void main(String[] args){
    String ficOrig, ficDest;//origine, destination
    Scanner lectClav = new Scanner(System.in);
    ficOrig = lectClav.nextLine();
    ficDest = lectClav.nextLine();
    if(ficOrig.equals(ficDest)){
          System.out.println("Impossible de copier
             un fichier dans lui même");
          return;
    }
    //ajouter le code pour copier (slide précédent)
}
```

Pour comparer deux `String`, appeler `equals(...)`, car `ficOrig==ficDest` compare les adresses mémoires.

Faire l'utilisateur saisir le fichier à copier

```
//n'oublier au début import java.util.*;
public static void main(String[] args){
    String ficOrig, ficDest;//origine, destination
    Scanner lectClav = new Scanner(System.in);
    ficOrig = lectClav.nextLine();
    ficDest = lectClav.nextLine();
    if(ficOrig.equals(ficDest)){
```

Remarquer que le code est plus propre avec les noms :

- `lectClav` ou lieu de `lecteurClavier`
- `ficOrig` ou lieu de `ficOrigine`
- `ficDest` au lieu de `ficDestination`

```
}
```

Pour comparer deux `String`, appeler `equals(...)`, car `ficOrig==ficDest` compare les adresses mémoires.

Et si on voulait rechercher un motif dans le texte ?

Une comande qui affiche toutes les lignes qui contient `toto` :

```
grep toto fichier.txt
```

Et sous Java :

```
BufferedReader lecteur
      = new BufferedReader(new FileReader("a.txt"));
String ligne = lecteur.readLine();
//readLine renvoie null à la fin du fichier
while(ligne!=null){
    if(ligne.indexOf("toto")>=0) //voir doc
        System.out.println(ligne);
    ligne = lecteur.readLine();
}
```

Chercher `String` `java` sur Internet pour la doc avec toutes les méthodes de la classe String.

Et si on voulait rechercher un motif dans le texte ?

Une comande qui affiche toutes les lignes qui contient `toto` :

```
grep toto fichier.txt
```

Et sous `Java` :

```
BufferedReader lecteur
      = new BufferedReader(new FileReader("a.txt"));
String ligne = lecteur.readLine();
//readLine renvoie null à la fin du fichier
while(ligne!=null){
    if(ligne.indexOf("toto")>=0) //voir doc
        System.out.println(ligne);
    ligne = lecteur.readLine();
}
```

Chercher `String java` sur internet pour la doc avec toutes les méthodes de la classe String.

Et si on voulait rechercher un motif dans le texte ?

Une comande qui affiche toutes les lignes qui contient `toto` :

```
grep toto fichier.txt
```

Et sous `Java` :

```
BufferedReader lecteur
      = new BufferedReader(new FileReader("a.txt"));
String ligne = lecteur.readLine();
//readLine renvoie null à la fin du fichier
while(ligne!=null){
    if(ligne.indexOf("toto")>=0) //voir doc
        System.out.println(ligne);
    ligne = lecteur.readLine();
}
```

 Chercher `String java` sur internet pour la doc avec toutes les méthodes de la classe String.

Introduction Collections Java

Le paquetage `java.util` fournit :

- grandes structures d'organisation de données (collections) : tableau de hachage (`Hashtable`), des tableaux, des listes triés, arbres, queues, piles, etc.
- des classes utilitaires pour gérer les chaînes (`StringTokenizer`), mais aussi pour gérer les dates et le temps

Documentation → `docs.oracle.com/javase/7/docs/api/java/util/package-summary.html`

Introduction Collections Java

Le paquetage `java.util` fournit :

- grandes structures d'organisation de données (collections) : tableau de hachage (`Hashtable`), des tableaux, des listes triés, arbres, queues, piles, etc.
- des classes utilitaires pour gérer les chaînes (`StringTokenizer`), mais aussi pour gérer les dates et le temps

Documentation → `docs.oracle.com/javase/7/docs/api/java/util/package-summary.html`

Exemple `Hashtable`

1. Trouver la documentation de la classe : chercher les mots clés “Hashtable javadoc”
2. Observer quelques méthodes : `put(clé,valeur)`, `get(clé)`, `containsKey(clé)`
3. Construire un objet `Hashtable` et utiliser ses méthodes

```
import java.util.*;
Hashtable agenda = new Hashtable();
agenda.put("Daniel","1234");
agenda.put("Claire","1234");
if (!agenda.containsKey("Daniel"))
    agenda.put("Daniel","9999");
System.out.println(agenda.get("Daniel"));
```

Exemple `Hashtable`

1. Trouver la documentation de la classe : chercher les mots clés "Hashtable javadoc"
2. Observer quelques méthodes : `put(clé,valeur)`, `get(clé)`, `containsKey(clé)`
3. Construire un objet `Hashtable` et utiliser ses méthodes

```
import java.util.*;
Hashtable agenda = new Hashtable();
agenda.put("Daniel","1234");
agenda.put("Claire","1234");
if (!agenda.containsKey("Daniel"))
    agenda.put("Daniel","9999");
System.out.println(agenda.get("Daniel"));
```

1. Lire un agenda à partir d'un fichier `agenda.txt` comme :

```
Ada    0675123456
Sonya  0623123456
Toto   0637123456
```

2. stocker tout dans un `Hashtable` (une Collection)
3. faire des opérations, ex, chercher le numéro de téléphone de Toto, chercher le le propriétaire d'un numéro, etc.

C'est une tache longue, on doit anticiper ce qu'on doit savoir faire

A Il faut savoir lire un fichier

B Il faut savoir couper une chaine en deux (proprio+num)

C Il faut savoir manipuler un `Hashtable`

On a vu les points `A` et `C`, on anticipe que le point B est plus difficile.

1. Lire un agenda à partir d'un fichier `agenda.txt` comme :

```
Ada    0675123456
Sonya  0623123456
Toto   0637123456
```

2. stocker tout dans un `Hashtable` (une Collection)
3. faire des opérations, ex, chercher le numéro de téléphone de Toto, chercher le le propriétaire d'un numéro, etc.

C'est une tache longue, on doit anticiper ce qu'on doit savoir faire

- A Il faut savoir lire un fichier
- B Il faut savoir couper une chaine en deux (proprio+num)
- C Il faut savoir manipuler un `Hashtable`

On a vu les points `A` et `C`, on anticipe que le point B est plus difficile.

1. Lire un agenda à partir d'un fichier `agenda.txt` comme :

```
Ada    0675123456
Sonya  0623123456
Toto   0637123456
```

2. stocker tout dans un `Hashtable` (une Collection)
3. faire des opérations, ex, chercher le numéro de téléphone de Toto, chercher le le propriétaire d'un numéro, etc.

C'est une tache longue, on doit anticiper ce qu'on doit savoir faire

A Il faut savoir lire un fichier

B Il faut savoir couper une chaine en deux (proprio+num)

C Il faut savoir manipuler un `Hashtable`

On a vu les points `A` et `C`, on anticipe que le point B est plus difficile.

Comment couper “abc def” en 2 chaînes ?

On cherche l’indice `espIndice` de l’espace, et on obtient :

- une chaîne `abc` à gauche de `espIndice`
- une chaîne `def` à droite.

```
class SplitStr{
    public static void main(String[] args){
        String x = "abc def";
        int espIndice = x.indexOf(" ");
        String abc = x.substring(0,espIndice);
        String def =
            x.substring(espIndice+1, x.length());
        System.out.println(abc);
        System.out.println(def);
    }
}
```

Comment couper “abc def” en 2 chaînes ?

On cherche l’indice `espIndice` de l’espace, et on obtient :

- une chaîne `abc` à gauche de `espIndice`
- une chaîne `def` à droite.

```
class SplitStr{
    public static void main(String[] args){
        String x = "abc def";
        int espIndice = x.indexOf(" ");
        String abc = x.substring(0,espIndice);
        String def =
            x.substring(espIndice+1, x.length());
        System.out.println(abc);
        System.out.println(def);
    }
}
```