

Algorithmique & Programmation

Extensions iJava : fichiers textuels

`yann.secq@univ-lille.fr`

ABDELKADER Omar, BIRLOUEZ Martin, BONEVA Iovka, DELECROIX Fabien, LEQUINIOU Erwann, MARSHALL-BRETON Christopher, REKIK Yosra, SECQ Yann, SOW Younoussa, SUDHEENDRAN Megha, SU Yue

Fonctions prédéfinies

Sorties (terminal)

```
void print(String s)
void print(int i)
void print(long l)
void print(float f)
void print(double d)
void print(char c)
void print(boolean b)
```

```
void println()
void println(String s)
void println(int i)
void println(long l)
void println(float f)
void println(double d)
void println(char c)
void println(boolean b)
```

Entrées (clavier)

```
int readInt()
long readLong()
float readFloat()
double readDouble()
String readString()
char readChar()
```

Sur les chaînes de caractères

```
boolean equals(String s1, String s2)
int compare(String s1, String s2)
int length(String s)
String substring(String s, int start, int end)
char charAt(String s, int idx)
String toUpperCase(String s)
String toLowerCase(String s)
```

Sur les nombres

```
double random()
int stringToInt(String s)
```

Sur les tableaux

```
int length(<tableau> t)
int length(<tableau> t,
            int dim)
```

Extensions iJava

- Comment utiliser d'autres fonctions prédéfinies, mais non disponibles par défaut ?
- Notion de bibliothèque (ou librairie, de l'anglais *library*)
- Ensemble de type et fonctions définies dans un autre espace de nommage ...
- Pour iJava, l'espace `extensions`
- Exemple d'usage avec la manipulation de fichiers

Extensions iJava

- Notion de fichier (cf. cours de système !)
- Gérer des fichiers textes
- Gérer des fichiers CSV
- Usages potentiels dans le cadre des projets

Notion de fichier

- Un ensemble de données ...
- Concept polymorphe avec UNIX !
- Limitons-nous aux fichiers textuels
- Stockage pérenne d'information textuelles
- Opérations : lecture et écriture

Notion de fichiers

- Deux approches possibles avec `iJava`
 - Lecture de chaînes de caractères ligne par ligne
 - Lecture sous forme de grille pour les fichiers de type Comma Separated Values (CSV)

Ligne par ligne

- Pas un type de base, nécessaire de l'**importer** explicitement !
- Utiliser le type `File`
- *Tant que « prêt », lire la prochaine ligne*
- Principales fonctions :

`extensions.File`

`newFile(String filename)`

`String`

`readLine(extensions.File file)`

`boolean`

`ready(extensions.File file)`

Réécrivons cat !

!! =>

```
import extensions.File;
class Cat extends Program {
    void algorithm() {
        final String FILENAME = "Cat.java";
        File f = new File(FILENAME);
        int nbLines = 0, nbChars = 0;
        while (ready(f)) {
            String currentLine = readLine(f);
            nbLines++;
            nbChars = nbChars + length(currentLine);
            println(currentLine);
        }
        println("-----");
        println("Le fichier "+FILENAME+" contient "+
            nbLines+" lignes et "+nbChars+" caractères.");
    }
}
```

Exécution de Cat

```
yannsecq@YANNs-MacBook-Pro fouillis % ijava execute Cat
```

```
import extensions.File;
class Cat extends Program {
    void algorithm() {
        final String FILENAME = "Cat.java";
        File f = new File(FILENAME);
        int nbLines = 0, nbChars = 0;
        while (ready(f)) {
            String currentLine = readLine(f);
            nbLines++;
            nbChars = nbChars + length(currentLine);
            println(currentLine);
        }
        println("-----");
        println("Le fichier "+FILENAME+" contient "+
            nbLines+" lignes et "+nbChars+" caractères.");
    }
}
```

```
-----
Le fichier Cat.java contient 17 lignes et 483 caractères.
```

Fichiers structurés : CSV

- Type de fichier permettant de représenter une base de données très simplifiée (une table)
- Fichier texte constitué de lignes dont les éléments sont séparés par des (points) virgules
- Utile pour sauvegarder des informations structurée sous forme de table de manière pérenne (ie. entre différentes exécutions)

Type fichier CSV

- Utiliser le type `CSVFile`
- S'apparente à un `String[][]` en mémoire
- Principales fonctions

<code>int</code>	<code>columnCount(CSVFile table)</code>
<code>int</code>	<code>columnCount(CSVFile table, int idxLine)</code>
<code>String[]</code>	<code>getAllFilesFromCurrentDirectory()</code>
<code>String[]</code>	<code>getAllFilesFromDirectory(String directory)</code>
<code>String</code>	<code>getCell(CSVFile table, int idxLine, int idxColumn)</code>
<code>CSVFile</code>	<code>loadCSV(String filename)</code>
<code>CSVFile</code>	<code>loadCSV(String filename, char separator)</code>
<code>void</code>	<code>saveCSV(String[][] content, String filename)</code>
<code>void</code>	<code>saveCSV(String[][] content, String filename, char separator)</code>

Fonctions: fichier CSV

```
// Charger un fichier CSV à l'aide de son nom de fichier  
CSVFile loadCSV(String filename)  
// Sauver un tableau à 2D de chaînes sous la format d'un  
// fichier CSV  
void saveCSV(String[][] content, String filename)  
  
// Nombre de ligne du fichier passé en paramètre  
int rowCount(CSVFile table)  
// Nombre de colonne du fichier passé en paramètre  
int columnCount(CSVFile table)  
// Idem mais si les colonnes varient selon les lignes (bof)  
int columnCount(CSVFile table, int idxLine)  
  
// Retourne la valeur contenue en (idxLine, idxColumn)  
String getCell(CSVFile table, int idxLine, int idxColumn)  
  
// Retourne un tableau avec la liste de l'ensemble des  
// fichiers contenus dans le répertoire courant  
String[] getAllFilesFromCurrentDirectory()
```

Manipulation de CSV

- Afficher le contenu des fichiers CSV du répertoire courant
- Lister tous les fichiers, filtrer sur l'extension, afficher le contenu d'un fichier
- Toujours séparer les traitements des affichages !

!! =>

```
import extensions.CSVFile;
class ListAllCSV extends Program {

    String[] getAllCSVFiles() {
        // à compléter
    }

    void print(CSVFile csv) {
        // à compléter
    }

    void algorithm() {
        // On récupère la liste de tous les fichiers
        String[] csvFiles = getAllCSVFiles();
        // Maintenant on parcourt chacun des fichiers
        // CSV et on affiche leur contenu
        for (int i=0; i<length(csvFiles); i++) {
            print(loadCSV(csvFiles[i]));
        }
    }
}
```

```
import extensions.CSVFile;
class ListAllCSV extends Program {

    String[] getAllCSVFiles() {
        ...
    }

    void print(CSVFile csv) {
        for (int line=0; line < rowCount(csv); line++) {
            for (int column=0; column < columnCount(csv, line); column++) {
                print(getCell(csv, line, column)+"|");
            }
            println();
        }
    }

    void algorithm() {
        String[] csvFileNames = getAllCSVFiles();
        // Maintenant on parcourt chacun des fichiers
        // CSV et on affiche leur contenu
        for (int i=0; i<length(csvFileNames);i++) {
            print(loadCSV(csvFileNames[i]));
        }
    }
}
```

```
String[] getAllCSVFiles() {
    String[] filenames = getAllFilesFromCurrentDirectory();
    String[] csvFiles = new String[length(filenames)];
    int idxCSV = 0;
    // On filtre les fichiers selon leur extension en
    // ne conservant que les fichiers ".csv"
    for (int idx=0; idx<length(filenames); idx++) {
        String filename = filenames[idx];
        String extension = "?";
        if (length(filename)>4) {
            extension = substring(filename, length
                                   (filename)-3,length(filename));
        }
        if (equals(extension, "csv")) {
            csvFiles[idxCSV] = filename;
            idxCSV = idxCSV + 1;
        }
    }
    // On recopie dans un tableau faisant juste la bonne taille
    String[] result = new String[idxCSV];
    for (int idx=0; idx<length(result); idx++) {
        result[idx] = csvFiles[idx];
    }
    return result;
}
```

Synthèse sur les fichiers

- Notion de librairie/bibliothèque et d'**importation**
- Accès/stockage d'informations pérennes
- Manipulation des fichiers textuels possible avec deux types différents en `iJava`
 - Soit ligne à ligne (`File`)
 - Soit sous forme de grille (`CSVFile`)
- **ATTENTION : ne pas oublier l'importation de ces types en tout début de programme !**



Tableau de promo

- Souhait d'un tableau d'étudiant ·e ·s (triés !) pour une année de BUT
- Les participant ·e ·s sont caractérisés par leur nom, prénom, numéro étu et une moyenne
- Souhait d'externaliser ces données afin qu'elles soient pérennes entre deux exécutions
- Comment procéder ?

Tableau de promo

- Toujours la même démarche
 - Analyser les données et les modéliser avec des types et structures de données
 - Analyser les traitements à réaliser pour évaluer la pertinence de la modélisation des données

Tableau de promo : traitements

- Principaux traitements sur les étudiants ?
 - Chargement d'un fichier contenant les données de l'ensemble d'une promotion
 - Charger ces données et les intégrer dans une structure de données pertinentes pour trier ensuite les étudiants sur les moyennes
 - Sauvegarder la SDD dans un (nouveau) fichier

Type Etudiant

```
class Etudiant {  
    String nom, prenom;  
    int numero;  
    double moyenne;  
}
```

// dans le fichier du programme principal

```
Etudiant newEtudiant(String nom, String prenom, int  
numéro,  
                        double moyenne) {  
    Etudiant e = new Etudiant();  
    e.nom = nom; e.prenom = prénom; e.numero = numéro;  
    e.moyenne = moyenne;  
    return e;  
}
```

```
String toString(Etudiant e) {  
    return nom+" "+prenom+" (" +numero+" ) - « +moyenne;  
}
```

Load and store

```
Etudiant[] load(String nomFichier) {  
    CSVFile etudiantsCSV = loadCSV(nomFichier);  
    Etudiant[] etudiants = new Etudiant[rowCount(etudiantsCSV)];  
    for (int idxLig=0; idxLig < length(etudiantsCSV); idxLig++) {  
        String nom      = getCell(etudiantsCSV, idxLig, 0);  
        String prenom    = getCell(etudiantsCSV, idxLig, 1);  
        int    numero    = stringToInt(getCell(etudiantsCSV, idxLig, 2));  
        double moyenne = stringToDouble(etudiantsCSV, idxLig, 3);  
        etudiants[idxLig] = newEtudiant(nom, prénom, numéro, moyenne);  
    }  
    return etudiants;  
}
```

Update

```
Etudiant trouver(Etudiant[] etudiants, int numéro) {  
    Etudiant étudiant = null;  
    int idx = 0;  
    while (idx < length(etudiants) && etudiants[idx].numero != numero) {  
        idx++;  
    }  
    if (idx < length(etudiants)) {  
        étudiant = etudiants[idx];  
    }  
    return étudiant;  
}
```

```
Etudiant trouver(Etudiant[] etudiants, int numéro) {  
    for (int idx=0; idx < length(etudiants); idx++) {  
        if (etudiants[idx].numero == numero) {  
            return etudiants[idx];  
        }  
    }  
    return null;  
}
```

Update

```
boolean setMoyenne(Etudiant[] etudiants, int numero,  
                   double moyenne) {  
    Etudiant cible = trouver(etudiants, numéro);  
    if (cible != null) {  
        cible.moyenne = moyenne;  
        return true;  
    }  
    return false;  
}
```

Save

```
void sauver(Etudiant[] etudiants, String nomFichier) {  
    String[][] etudiantsCSV =  
        new String[length(etudiants)][4];  
    // initialisation du tableau de chaînes  
    for (int idxL=0; idxL < length(etudiants); idxL++) {  
        etudiantsCSV[idxL][0] = etudiants[idxJ].nom;  
        etudiantsCSV[idxL][1] = etudiants[idxJ].prenom;  
        etudiantsCSV[idxL][2] = etudiants[idxJ].numero;  
        etudiantsCSV[idxL][3] = etudiants[idxJ].moyenne;  
    }  
    // sauvegarde du tableau sous forme de CSV  
    saveCSV(etudiantsCSV, nomFichier) ;  
}
```

Conclusion

- Retenir la démarche !
- Analyser les données et les modéliser avec des types et structures de données
- Analyser les traitements à réaliser pour évaluer la pertinence de la modélisation des données
- Exemple proche de vos besoins pour la SaÉ2 !

