

Info 2 SCP

Les fichiers

Damien Sauveron

<http://damien.sauveron.fr/>

Problématiques

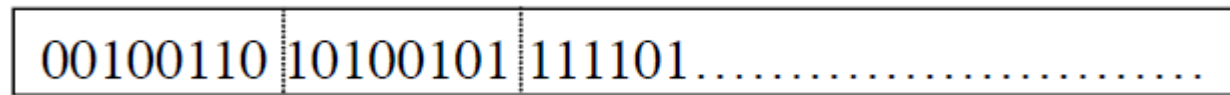
- Jusqu'à présent nous avons été capables de produire des informations, des résultats, etc. mais nous ne pouvions pas les conserver pour une exploitation ultérieure.
 - Les fichiers vont répondre à cette problématique en nous permettant de stocker nos données
- Jusqu'à présent nous nous sommes souvent limités à des programmes simples, ou tout au moins nous nous sommes restreints à des cas simples puisque les seules données d'entrée dont nous disposions étaient celles saisies par l'utilisateur (ou celles générés aléatoirement).
 - Les fichiers vont répondre à cette problématique en nous permettant d'accéder à de nouvelles sources de données.

Définition et caractéristique d'un fichier

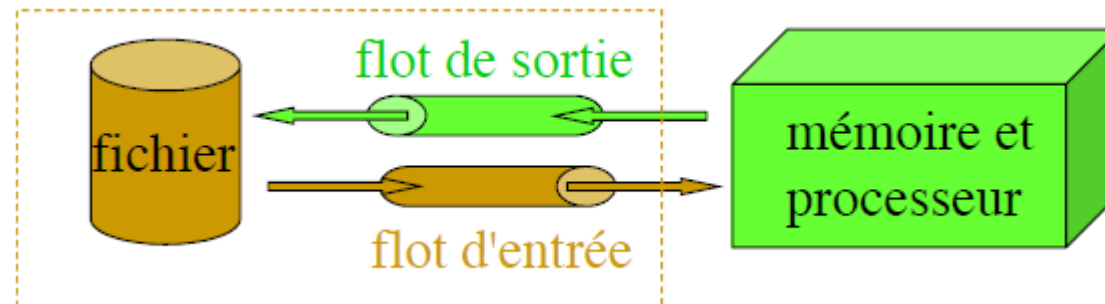
- Un fichier est une collection de données numériques réunies sous un même nom, enregistrées sur une mémoire secondaire (disque dur, clé USB, CD-ROM, disquette, ...).
 - Souvent le nom du fichier comprend une extension précisant le type d'information qu'il contient :
 - .txt pour des informations textuelles ;
 - .png pour des images ;
 - ...
- Ces supports mémoire disposent généralement de grandes capacités de stockage par rapport à la mémoire vive.
 - En To pour les supports secondaires contre des Go pour la mémoire vive
- Ces supports mémoire sont aussi de type persistant
 - En l'absence d'alimentation les données contenues sont conservées (contrairement à la mémoire vive).
- Ces supports mémoire sont souvent plus lents en accès que la mémoire vive.

Fichier (vue physique)

- Sur le support mémoire, le fichier peut être vu comme une suite de bits.

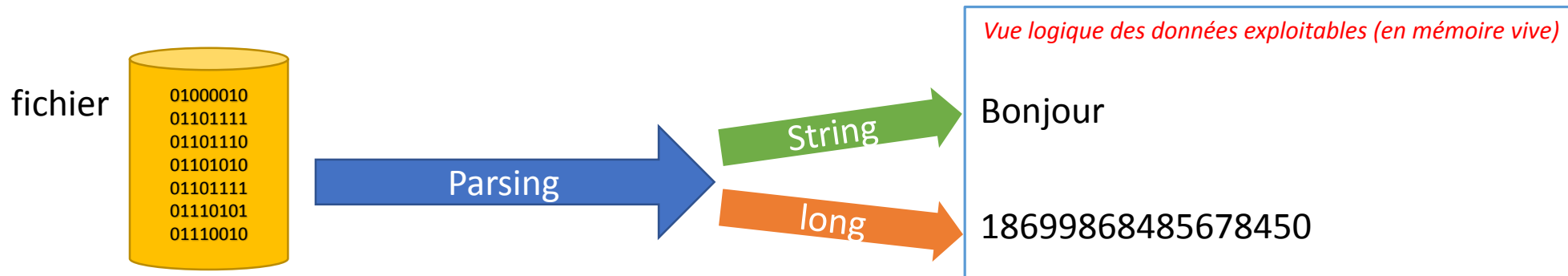


- Les fichiers peuvent être considérés comme des canaux d'entrées et de sorties utilisant des flots de données :
 - réception de données dans le cas d'un flot de sortie (comme l'affichage à l'écran) ;
 - émission de données dans le cas d'un flot d'entrée (comme la saisie au clavier).



Fichier (vue logique)

- Pour être exploitable les données brutes d'un fichier doivent être « parsées », c'est-à-dire analysées et transformées en informations utilisables.



- Le parsing peut être défini comme l'opération d'extraire une information structurée d'un flux de données.
- Un ensemble de « caractères » informatiques d'un côté, une information utilisable de l'autre, et au milieu analyse et extraction.
- On peut bien sûr parser les données pour obtenir soit des informations de **types de base**, soit **de types composés**.

Les fichiers en Processing

- Selon les langages de programmation utilisés, deux grands types de fichiers peuvent être utilisés : les fichiers texte et les fichiers binaire
- Si les fichiers binaire sont plus compacts et offrent plus de possibilités, pour notre usage (et surtout le votre), les fichiers texte seront suffisants.
- En Processing, nous nous restreindrons donc à l'utilisation de fichiers texte
 - Non pas qu'il soit impossible de manipuler des fichiers binaire
 - C'est possible avec certaines fonctions, mais ce n'est pas très pratique.
- Attention, un fichier texte n'est pas forcément un fichier avec l'extension .txt.
 - C'est un fichier qui lorsqu'on l'ouvre dans un éditeur de texte a du sens.
 - Du sens = qu'il est lisible est compréhensible !
 - Un fichier csv, un fichier html, un fichier txt sont des fichiers compréhensibles
 - Un fichier exe, un fichier dll, un fichier .so sont des fichiers incompréhensibles

Générer un fichier texte : `PrintWriter`

- Pour écrire dans un fichier, il faut créer un écrivain (*Writer* en anglais) avec la fonction `createWriter` qui crée le fichier et l'ouvre en écriture.
 - On notera qu'un écrivain est une variable de type `PrintWriter`
- Utiliser la fonction `println` de cet écrivain en lui fournissant les données à écrire.
- Fermer le fichier lorsque l'on a fini de l'utiliser avec la fonction `close`.
 - On pensera simplement à demander l'écriture effective de toutes les données en attente au travers de la fonction `flush`.

Écrire dans un fichier `dates.txt`

```
class Date
{
    int jour;
    int mois;
    int annee;
    Date(int j, int m, int a) {
        jour = j; mois = m; annee = a;
    }

    Date(Date d) {
        jour = d.jour;
        mois = d.mois;
        annee = d.annee;
    }
}

void afficheDate(Date d) {
    println(d.jour + "/" + d.mois + "/" + d.annee);
}

void ecrireDate(Date d, PrintWriter o) {
    o.println(d.jour + "," + d.mois + "," + d.annee);
}
```

```
void setup() {
    PrintWriter out;

    out = createWriter("dates.txt");

    Date d1 = new Date(23, 1, 2017);
    Date d2 = new Date(24, 1, 2017);
    afficheDate(d1); // affiche 23/1/2017
    afficheDate(d2); // affiche 24/1/2017

    ecrireDate(d1, out);
    ecrireDate(d2, out);

    out.flush();
    out.close();
}
```

- Créer le fichier `dates.txt` avec le contenu (si le fichier existe déjà il l'écrase le contenu !) :

```
23,1,2017
24,1,2017
```

Lire un fichier texte : `BufferedReader` méthode 1

- Pour lire un fichier, il faut créer un lecteur (*Reader* en anglais) avec la fonction `createReader` qui ouvre le fichier en lecture.
 - On notera qu'un lecteur est une variable de type `BufferedReader`
- Utiliser la fonction `readLine` de ce lecteur pour récupérer les données du fichier ligne par ligne.
 - Si la valeur retournée vaut `null` alors on a fini de lire le fichier
- Pour chaque ligne non `null`, il convient de « parser » les données pour les mettre au format voulu avec la fonction `split`
- Fermer le fichier lorsque l'on a fini de l'utiliser avec la fonction `close`.
- Attention, il faut gérer les erreurs avec un bloc

```
try {  
    } catch (IOException e) {  
    }
```

Lire un fichier dates.txt

```
class Date
{
    int jour;
    int mois;
    int annee;
    Date(int j, int m, int a) {
        jour = j; mois = m; annee = a;
    }

    Date(Date d) {
        jour = d.jour;
        mois = d.mois;
        annee = d.annee;
    }
}

void afficheDate(Date d) {
    println(d.jour + "/" + d.mois + "/" + d.annee);
}

void ecrireDate(Date d, PrintWriter o) {
    o.println(d.jour + "," + d.mois + "," + d.annee);
}
```

```
Date lireDate(String s) { // Pour parser la ligne
    int j, m, a;
    String[] jma = split(s, ","); // découpe selon les ,
    j = int(jma[0]);
    m = int(jma[1]);
    a = int(jma[2]);

    return new Date(j,m,a);
}

void setup() {
    try {

        BufferedReader in;

        in = createReader("dates.txt");

        String line;
        Date d;
        do {
            line = in.readLine();
            if (line != null) {
                d = lireDate(line);
                afficheDate(d);
            }
        } while (line != null);

        in.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

- Lit le contenu du fichier dates.txt et affiche :

23/1/2017
24/1/2017

Lire un fichier texte : `loadStrings()` méthode 2

- Pour lire la totalité des lignes d'un fichier texte, il est possible d'utiliser la fonction `loadStrings()`.
- Pour chaque ligne lue, il convient de « parser » les données pour les mettre au format voulu avec la fonction `split`

Lire un fichier dates.txt

```
class Date
{
    int jour;
    int mois;
    int annee;
    Date(int j, int m, int a) {
        jour = j; mois = m; annee = a;
    }

    Date(Date d) {
        jour = d.jour;
        mois = d.mois;
        annee = d.annee;
    }
}

void afficheDate(Date d) {
    println(d.jour + "/" + d.mois + "/" + d.annee);
}

void ecrireDate(Date d, PrintWriter o) {
    o.println(d.jour + "," + d.mois + "," + d.annee);
}
```

```
Date lireDate(String s) { // Pour parser la ligne
    int j, m, a;
    String[] jma = split(s, ","); // découpe selon les ,
    j = int(jma[0]);
    m = int(jma[1]);
    a = int(jma[2]);

    return new Date(j,m,a);
}

void setup() {

    String[] lines = loadStrings("dates.txt");

    Date d;
    for (String line : lines) {
        d = lireDate(line);
        afficheDate(d);
    }
    // ou à l'ancienne
    //for (int i = 0; i < lines.length; i++) {
    //    d = lireDate(lines[i]);
    //    afficheDate(d);
    //}
}
```

- Lit le contenu du fichier dates.txt et affiche :

```
23/1/2017
24/1/2017
```

Lire un fichier texte : `loadStrings()`

méthode 2 - URL

- On notera qu'il est même possible d'utiliser la fonction `loadStrings()` pour lire des fichiers via des URL (Uniform Resource Locator).
- Ainsi pour lire la page <http://processing.org/about/index.html> ou mieux <http://damien.sauveron.fr/>, il suffit de faire :

```
void setup() {  
  
    String[] lines = loadStrings("http://damien.sauveron.fr/");  
  
    println("Il y a " + lines.length + " lignes sur cette page.");  
    for (String line : lines) {  
        println(line);  
    }  
}
```

Ajouter des données à un fichier texte existant

- Pour ajouter des données à un fichier déjà existant, on lira son contenu en totalité puis on écrira à nouveau ce contenu avant d'ajouter les nouvelles données à la fin du fichier.
- Bien sûr, il est possible d'ajouter des données à une autre position que la fin.
 - Il faut alors insérer celles-ci aux positions souhaitées avant de réécrire dans le fichier.

```
void setup() {  
  
    String[] lines = loadStrings("dates.txt");  
    PrintWriter out;  
    out = createWriter("dates.txt");  
  
    for (String line : lines) {  
        out.println(line);  
    }  
  
    // Ajout d'une nouvelle date  
    Date d = new Date(25, 1, 2017);  
    ecrireDate(d, out);  
  
    out.flush();  
    out.close();  
}
```

Attention ce n'est pas la solution la plus efficace mais c'est la plus simple avec ce que nous avons vu !

Modifier les données d'un fichier texte existant

- Il est aussi évidemment possible de modifier des données existantes.
 - Il suffit d'opérer les modifications avant de réécrire ces données dans le fichier.

```
void setup() {  
    String[] lines = loadStrings("dates.txt");  
    PrintWriter out;  
    out = createWriter("dates.txt");  
  
    Date d;  
    for (String line : lines) {  
        d = lireDate(line);  
  
        if (d.jour == 24) {  
            d.annee = 2018;  
        }  
  
        ecrireDate(d, out);  
    }  
  
    out.flush();  
    out.close();  
}
```

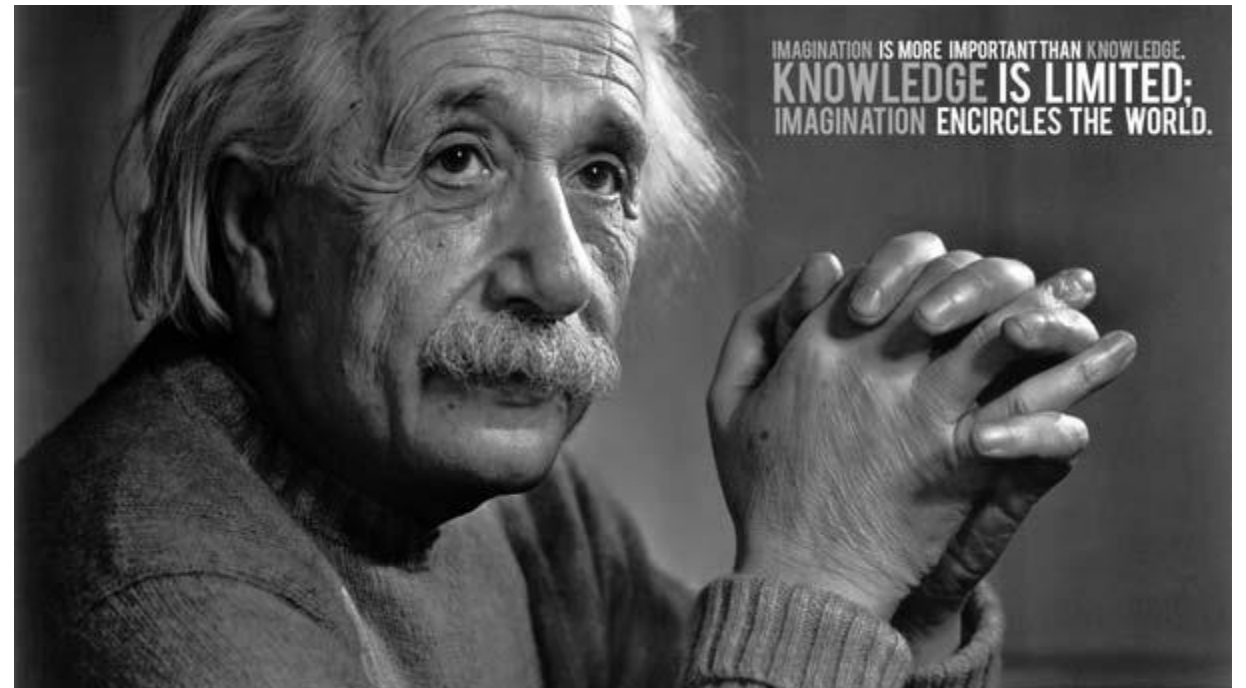
Et maintenant

- *Et maintenant que vais-je faire*
- *De tout ce temps que sera ma vie*
- ...

Gilbert Bécaud

- Par exemple :
 - Produire des fichiers de données que je peux analyser sous Excel (csv) ou sous d'autres outils.
 - Alimenter mes programmes avec des données provenant de programmes externes.

- Mais vous pouvez aussi :
 - produire des animations,
 - produire de fichiers PDF,
 - ...



Réalisation d'une petite animation – Partie 1 (enregistrement d'un mouvement)

```
class Point
{
    int x;
    int y;
    Point(int x, int y) {
        this.x = x;  this.y = y;
    }

    Point(Point p) {
        x = p.x;
        y = p.y;
    }
}

void ecrirePoint(Point p, PrintWriter o) {
    o.println(p.x + "-" + p.y);
}
```

```
PrintWriter out;
Point p;

void setup() {
    out = createWriter("positions.txt");

    size(600,400);
    background(255);
    // on calibre une vitesse d'exécution
    frameRate(24);
}

void draw() {
    // affiche 1 point
    point(mouseX, mouseY);

    p = new Point(mouseX, mouseY);
    ecrirePoint(p, out);
}

void keyPressed() {
    out.flush();
    out.close();
    exit(); // Arrête le programme
}
```

Si besoin on peut *nettoyer* le fichier
positions.txt avec un éditeur de texte.

Réalisation d'une petite animation – Partie 2 (lecture d'un mouvement)

```
class Point
{
    int x;
    int y;
    Point(int x, int y) {
        this.x = x;  this.y = y;
    }

    Point(Point p) {
        x = p.x;
        y = p.y;
    }
}

Point lirePoint(String s) {
    String[] coord = split(s,"-");
    return new Point(int(coord[0]), int(coord[1]));
}

void ecrirePoint(Point p, PrintWriter o) {
    o.println(p.x + "-" + p.y);
}

void affichePoint(Point p) {
    ellipse(p.x,p.y,20,20);
    // rect(p.x,p.y,2,10);
}
```

```
String[] lines;
Point p;
int compteur;

void setup() {
    lines = loadStrings("positions.txt");
    size(600,400);
    background(255);
    frameRate(24);
    compteur = 0;
}

void draw() {
    p = lirePoint(lines[compteur]);
    // affiche notre point
    affichePoint(p);

    compteur++;
    if(compteur > lines.length-1){
        saveFrame("resultat.jpg");
        noLoop();
    }
}

void keyPressed() {
    compteur=0;
    background(255);
    loop();
}
```

Réalisation d'une petite animation – Partie 3 (et pourquoi pas lire à l'envers)

```
class Point
{
    int x;
    int y;
    Point(int x, int y) {
        this.x = x;  this.y = y;
    }

    Point(Point p) {
        x = p.x;
        y = p.y;
    }
}

Point lirePoint(String s) {
    String[] coord = split(s, "-");
    return new Point(int(coord[0]), int(coord[1]));
}

void ecrirePoint(Point p, PrintWriter o) {
    o.println(p.x + "-" + p.y);
}

void affichePoint(Point p) {
    ellipse(p.x,p.y,20,20);
    // rect(p.x,p.y,2,10);
}
```

```
String[] lines;
Point p;
int compteur;

void setup() {
    lines = loadStrings("positions.txt");
    size(600,400);
    background(255);
    frameRate(24);
    compteur = lines.length-1;
}

void draw() {
    p = lirePoint(lines[compteur]);
    // affiche notre point
    affichePoint(p);

    compteur--;
    if(compteur < 0){
        saveFrame("resultat.jpg");
        noLoop();
    }
}

void keyPressed() {
    compteur = lines.length-1;
    background(255);
    loop();
}
```

Pour finir en beauté : PDF

```
import processing.pdf.*;

PGraphicsPDF pdf;

void setup(){
  size(594, 842);
  background(255);
  pdf = (PGraphicsPDF)beginRecord(PDF, "coolpdf.pdf"); // on démarre l'écriture
}

void draw(){
  float taille=random(60);
  ellipse(mouseX, mouseY, taille, taille); // dessiner des cercles de taille aléatoire à l'endroit de la souris
}

void keyPressed() {
  switch (key) {
    case ' ': // barre d'espace : page suivante
      pdf.nextPage(); // créer une nouvelle page
      background(255);
      break;
    case 'x': // x : pdf terminé
      endRecord();
      exit(); // Quitter pour être sur que tout est ok
      break;
  }
}
```