Génération de planning Latex

Licence info S5
TP COMPIL – 2011 - 2012

Le but de ce TP est de vous faire développer une petite application de traitement de données textuelles sans utiliser les techniques qui seront vues en cours, pour bien vous faire prendre la mesure de la difficulté de la tâche.

1 Le problème

Le but est de générer des affiches pour les soutenances des M2, au format latex compilé en pdf. Partant d'un fichier textuel décrivant le planning de l'ensemble des masters et couvrant toutes les journées de soutenance, on veut générer un fichier latex par jour et par master. Le nom du fichier incluera le nom du master, la date et la salle. Pour ce faire, on décrit le planning en utilisant un petit DSL maison.

1.1 Le DSL

Pour se simplifier la vie, et comme on ne sait pas encore utiliser une analyse dirigée par la syntaxe [1], on a fait le choix d'une analyse dirigée par les délimiteurs [2], en l'occurrence les sauts de ligne. On impose donc dans la syntaxe du DSL que les informations soient organisées ligne par ligne (et le fichier d'entrée sera en conséquence lu ligne par ligne, et non caractère par caractère). On pourra utiliser le saut de ligne et les blancs à volonté, pour la mise en forme.

Le DSL existe en 3 versions. C'est la version la plus complète qui a été utilisée en pratique pour les soutenances de septembre 2011. Pour ce TP, c'est déjà bien si vous réalisez la version simple. Les versions plus avancées sont à la fin du sujet.

1.1.1 Version simple : une seule journée

Le planning d'une journée commence par un en-tête décrivant le nom du master, la date et la salle. L'en-tête a la syntaxe suivante :

1. la première ligne commence par le mot clé `master`, suivi du nom du master (dans le cas du FIL : `IAGL`, `TIIR`, `eServices` et `IVI`, mais on peut accepter n'importe quelle suite de lettres) ;
2. la seconde ligne décrit la date : mot clé `date` suivi de la date écrite en toute lettre ou en abrégé ;
3. la troisième ligne décrit la salle : mot clé `salle` suivi d'une suite de lettres et/ou chiffres.

On trouve ensuite le planning proprement dit : la description de chaque soutenance, avec une soutenance par ligne. Sur une ligne de soutenance, les informations sont séparées par le délimiteur "`;`". Elles n'ont pas de format particulier et seront recopiées telles quelles dans le code Latex généré. Les informations sont les suivantes : description du créneau horaire, puis nom et prénom de l'étudiant, puis nom de l'entreprise, et enfin mots clés et commentaires qui décrivent la soutenance. Les informations peuvent être manquantes, les séparateurs "`;`" demeurent.

Voici un exemple de planning pour une journée.

```
master TIIR
date 8 septembre 2011
salle A40

8h30-9h15 ; ; ;
9h15-10h00 ; Dupont Martin ; NEF ; J2EE, Scrum, DD
10h00-10h45 ; Nguyen Julie ; Enercoop ; php, cahier des charges
10h45-11h30 ; El Amrani Hassan ; Crédit Coopératif ; huis-clos
```

1.2 L'affichage souhaité

Le nom du fichier latex généré devra être de la forme `nomMaster_date_salle.tex`, ce qui produira finalement un fichier `nomMaster_date_salle.pdf`.

Pour chaque pdf généré, on souhaite faire apparaître (c'est expliqué plus loin comment) :

1. Analyse dirigée par la syntaxe = basée sur une grammaire algébrique
2. Delimiter-directed translation, [Fowler2011] p201.

- sous la forme d'un titre Latex : la mention « Soutenances des master2 », le nom du master, la date, la salle ;
- sous la forme d'un tableau Latex à 4 colonnes : le planning, à raison d'une ligne de tableau par ligne de soutenance.

Des `pdf` sont fournis à titre d'exemple dans le répertoire `test`.

1.3 Quelques mots à propos de Latex

Votre moulinette génère un fichier Latex, sans doute assez incompréhensible pour un non initié. Néanmoins pas besoin d'être un expert Latex pour réaliser ce TP.

- les morceaux de texte Latex à écrire dans le fichier vous sont donnés sous forme de constantes de type `String` déclarées statiquement dans le fichier `ConstantesLatex.java`
- pour voir à quoi ça ressemble, appeler `pdflatex <fichier.tex>` qui produit, entre autre fichiers auxiliaires (que vous pouvez effacer), un pdf que vous pouvez afficher (par exemple avec `evince`)
- votre enseignant de TP vous aidera à comprendre les messages d'erreur de `pdflatex` si besoin est.

Le texte qui suit se réfère aux constantes statiques de `ConstantesLatex.java`.

Un document Latex commence par une clause `documentclass` et des import de paquetage : ils sont groupés dans la constante `DEBUT_DOCUMENT`. Ensuite on trouve dans l'ordre :

- des déclarations de commande Latex, qu'on peut voir comme des macros, définissant par exemple la commande `\lemaster` valant `TIIR`. Il y a une commande `\lemaster`, une commande `\lasalle`, et une commande `\ladate`, à définir en utilisant les informations extraites du fichier d'entrée. Pour générer le code Latex :
  `\newcommand{\lemaster}{TIIR}`
  on concaténera la constante `DEBUT_COMMANDE_MASTER`, la chaîne `"TIIR"`, puis la chaîne `FIN_COMMANDE`
- la définition du titre Latex : constante `DEFINITION_TITRE`

Ensuite vient le (ou les) tableau(x) contenant les lignes de soutenances :

- la déclaration du tableau se fait par la constante `DEBUT_TABLEAU` ;
- ensuite viennent les lignes du tableau : le contenu des cases est écrit à la suite, la constante `SEPARATEUR_CASE` permettant de séparer les cases, et chaque ligne est terminée par la constante `FIN_LIGNE_TABLEAU` ;
- on termine le tableau par `FIN_TABLEAU`

On termine le document Latex par la constante `FIN_DOCUMENT` (sans oublier de fermer le fichier physique).

2 Réalisation

Télécharger l'archive sur le portail. Elle contient un paquetage `planningMaster` à compléter. Générer la javadoc peut être utile (`sh genererJavadoc.sh`).

2.1 Les fichiers fournis

La classe `GenerationLatexPlanningMaster` est à compléter. La méthode `genererUnPlanning` est chargée de :

- lever une exception `ErreurFormatException` si le fichier d'entrée n'est pas correct ;
- s'il est correct, écrire le fichier Latex correspondant.

Les utilitaires d'entrée/sortie sont fournis et paramètrent la classe `GenerationLatexPlanningMaster` :

- `LectureLigne.java`, `LectureLigneFlot.java` : pour lire un fichier ligne par ligne ;
- `EcritureTexte.java`, `EcritureTexteFlot.java` : pour écrire des chaînes dans un fichier.

Comme le nom du fichier de sortie n'est connu qu'une fois le début de l'analyse commencée, l'objet de type `EcritureTexteFlot` n'est pas passé à l'objet `GenerationLatexPlanningMaster` via son constructeur, mais via l'accesseur `setOutput`.

La classe `LancerGeneration` permet de lancer l'application en passant un fichier de données `fichier.dsl` sur la ligne de commande. Elle crée un objet de type `GenerationLatexPlanningMaster` et lance la méthode `genererUnPlanning`.

Le répertoire `test` contient des fichiers d'exemple, et les fichiers `tex` et `pdf` correspondant.

2.2 Méthode à appliquer

Le schéma général consiste à :
- lire la prochaine ligne
- la reconnaître pour vérifier qu'elle correspond à la syntaxe attendue, ou pour faire un choix entre plusieurs syntaxes possibles
- l'analyser pour en extraire l'information utile

Pour savoir quel formalisme utiliser pour la reconnaissance, on peut commencer par se demander si les langages contenant les lignes déclarant le master, la date, une ligne de soutenance, etc, sont des langages réguliers (avec réponse argumentée, pas au pif!).

Si tel est le cas, alors chaque ligne peut être reconnue par une expression régulière et on peut utiliser les expressions régulières de Java [3].

2.2.1 Les expressions régulières de Java

Vous trouverez ici un tutorial très complet (le chercher en local au M5)
`http://download.oracle.com/javase/tutorial/essential/regex/`
et dans ce sujet de quoi commencer. On se sert des classes `Pattern` et `Matcher` du paquetage `java.util.regex`.

La classe `Pattern` permet de définir une expression régulière sous la forme d'une `String`. Par exemple, `"master"` correspond à l'expr reg `master`. De nombreuses classes de caractères sont prédéfinies, voir
`http://download.oracle.com/javase/tutorial/essential/regex/pre_char_classes.html`
Par exemple, `\w` désigne n'importe quel caractère dans `[a-zA-Z_0-9]` et `\s` désigne un blanc au sens large (incluant les tabulations, retour à la ligne, etc). L'expression régulière `.` dénote n'importe quel caractère. L'expression `[^;]` désigne n'importe quel caractère sauf `;`. Les expressions `\s+` et `\s*` utilisent les opérateurs `+` et `*` avec leur signification habituelle. Comme en Java le caractère `\` est un caractère spécial, il faut le déspécialiser à l'intérieur des `String`, et on devra donc écrire `"\\s*"`, ce qui rend les expressions rapidement illisibles.

Si on veut juste savoir si une chaîne correspond à une expression régulière, on peut utiliser la méthode statique `matches` : `Pattern.matches("master",ligneLue)` retourne vrai si la chaîne d'entrée `ligneLue` (de type `String`) correspond à l'expression régulière `"master"`.

Pour extraire des sous-motifs de la chaîne d'entrée, il faut passer par la classe `Matcher`.

```
Pattern pattern = Pattern.compile("\\s*dring\\s+(\\w+)\\s*");
Matcher matcher = pattern.matcher(ligneLue);
if (matcher.matches())
    System.out.println(matcher.group(1));
```

définit l'expression régulière reconnaissant le mot `dring`, suivi d'une suite de caractères de `[a-zA-Z_0-9]`, avec éventuellement des blancs qui trainent. L'objet `matcher` est construit sur cette expression. Sa méthode `matches` retourne vrai si la chaîne d'entrée correspond à l'expression. L'appel de `matches` permet d'appeler la méthode `group`. Un groupe est une expression régulière entre parenthèses, la totalité de l'expression étant par convention le groupe 0. Les groupes sont numérotés par l'ordre d'apparition dans la chaîne de leur parenthèse ouvrante. Dans notre exemple, le groupe qui nous intéresse est `(\\w+)`, c'est le groupe numéro 1. Si `ligneLue` vaut `" dring il est 6h "`, alors le code affichera `il est 6h`.

2.3 Travail à rendre

À l'issue des 2 séances de TP, vous devez rendre sur PROF une archive de votre répertoire de travail, travail qui aura été abondamment testé à la fois dans les cas corrects (génération d'un fichier latex) et incorrect (levée d'exception). Vous rendrez aussi un `README` indiquant l'état d'avancement de votre travail (un travail bien mais pas complètement fait est mieux noté qu'un travail tout mais mal fait) et répondant aux questions suivantes.

Concernant le langage utilisé pour décrire les plannings : est-il possible d'utiliser dans la partie « mots clé ou commentaire sur la soutenance » (4ème colonne du tableau généré) :

3. Plus généralement, le langage contenant l'ensemble des plannings est régulier. Mais ce n'est pas pratique d'utiliser une grosse expression régulière pour extraire les informations du texte source.

– le mot `planning`
– le caractère `;`

En ce qui concerne l'approche « lire ligne par ligne et se débrouiller avec les expressions régulières Java » utilisée dans ce TP. À votre avis :

1. est-il facile de modifier l'un des mots clés ? (ex : remplacer "master" par "diplome")
2. est-il facile de modifier la syntaxe des plannings ? (ex : autoriser les déclarations de master, salle et date dans n'importe quel ordre)
3. est-il facile de modifier l'application pour générer un autre format que du Latex ?
4. en lisant votre code, est-il facile de repérer le lexique et la syntaxe du DSL ?
5. en cas d'erreur de lexique ou de syntaxe, le message produit par votre application permet-il de retrouver facilement l'erreur ?
6. si la réponse aux questions précédentes est parfois non : pensez-vous que c'est l'approche qui est en cause, ou votre code qui n'est pas clair ?

Les fichiers README et GenerationLatexPlanningMaster.java devront contenir votre nom.

3 Les versions complètes

3.1 Au niveau du DSL

3.1.1 Version avec pause

On a vite besoin d'enrichir le langage pour introduire des espacements sur l'affiche, correspondant aux pauses. Comme il y a des petites pauses café et de grandes pauses repas, on introduit deux nouveaux mots clés : `petitePause` et `grandePause`. Ces mots clés apparaissent (ou non) entre les lignes de soutenances, à raison d'un mot clé par ligne. Voir l'exemple `exempleAvecPause.dsl` dans le répertoire `test`.

3.1.2 Version complète

Ultimement, le fichier de description contient tous les plannings de toutes les journées, à la suite les uns des autres. Voir le fichier `exempleAvecPauseEtPlusieursPlannings.dsl` dans le répertoire `test`.

3.2 Au niveau de l'affichage et du Latex

Dans les tableaux générés :
– une petite pause se traduit par un petit espacement entre lignes : constante `PETITE_PAUSE` ;
– une grande pause se traduit par la fermeture du tableau courant et la réouverture d'un nouveau tableau : constante `GRANDE_PAUSE`.

L'enchaînement de plusieurs plannings se traduit par la fermeture du document Latex courant, et l'ouverture d'un autre document.