L3-CPOO 2007/08
Sujet de projet

Projet N°2 de CPOO : Java to Lisaac - Convertisseur automatique

Travail demandé

Le travail demandé comporte une modélisation UML et une implantation en Java ou Lisaac. Un rapport court (2 ou 3 pages) devra accompagner les diagrammes UML en format papier. Ce rapport devra

- expliquer vos choix de modélisation et lever les ambiguïtés du sujet
- présenter la répartition du travail

Modélisation UML

Le nombre et le type de diagrammes UML à concevoir est laissé à votre appréciation, mais cet ensemble de diagrammes devra d'une part modéliser correctement le système décrit ci-dessous, et d'autre part être aisément compréhensible par le correcteur. Votre modélisation UML devra comporter au minimum :

- Les diagrammes d'heritages
- Les diagrammes des classes
- Un diagramme d'états-transitions

Implantation

- Votre programme devra utiliser le langage Java ou Lisaac (au choix), et devra fonctionner sur la machine *turing*.
- L'interface utilisateur est textuelle (ligne de commande).
- Une petite explication de l'utilisation de votre programme est aussi nécessaire.

Remise du projet

La remise du projet se fera en deux temps :

- Le rapport et la modélisation UML du projet est à rendre **sur papier** lors de la dernière séance de TD (**semaine du 26 Novembre**).
- La programmation qui découle de votre modélisation est **à rendre pour le 04 Janvier**. La remise de cette partie du projet se fera électroniquement par mail à votre responsable de TP, à savoir :
  - L3-PF G1 & G3, L3-RIA : Benoit Sonntag (`sonntag@icps.u-strasbg.fr`)
  - L3-PF G2 & G4 : Claire Baegert (`baegert@dpt-info.u-strasbg.fr`)

Réalisation du projet

- La réalisation de ce projet devra se faire impérativement **par groupe de deux ou trois** pour que vous vous répartissiez le travail au sein du groupe.
- Comme tout cahier des charges, celui-ci ne peut être exhaustif. En cas d'ambiguïté, préciser votre interprétation personnelle, et éventuellement les questions à poser à votre interlocuteur (responsable de projet, futurs utilisateurs, etc.). Toute solution cohérente, justifiée et non contradictoire avec le cahier des charges sera acceptée.

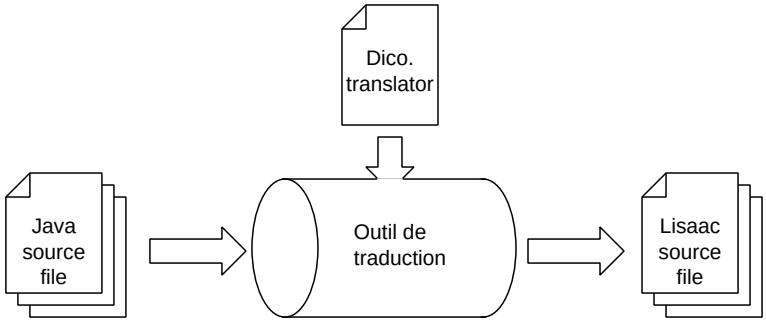
Sujet

Dans le cadre du développement d'un nouveau langage de programmation objet, nous vous proposons d'élaborer un outil permettant la traduction la plus automatique possible d'un source écrit en Java vers un source écrit en Lisaac.

Un nombre impressionnant d'applications et librairies sont disponible en Java, votre travail permettrait de récupérer avec un minimum d'efforts l'ensemble de ces sources sous Licence libre pour les intégrer au projet Lisaac.

Les noms des classes (ou prototypes) et des méthodes de la librairie standard du Lisaac n'étant pas les mêmes en Java, il faut aussi prévoir un dictionnaire de conversion des noms. Ce n'est pas a vous de remplir ce dictionnaire, mais simplement de le gérer correctement.

Le graphique suivant illustre l'outil en question :



1 Production du fichier de sortie

Une classe doit donner lieu à un fichier. Le nom du fichier produit doit être en minuscule avec l'extension `.li`.

2 Informations utiles entre Java et Lisaac

2.1 Les briques de base du Java

Java	Lisaac
`if (<condition>) <instr>`	`(<condition>).if { <instr> }`
`if (<condition>) <instr1> else <instr2>`	`(<condition>).if { <instr1> } else { <instr2> }`
`while (<condition>) <instr>`	`{ <condition> }.while_do { <instr> }`
`do <instr> while (<condition>)`	`{ <instr> }.do_while { <condition> }`
`switch <expr> { <case> }` `case <cste> : <instr> break ;`	`<expr>` `.when <cste> then { <instr> }`

Attention : Conservez les commentaires dans votre traduction.

2.2 L'heritage

L'équivalent à un héritage de classe en Lisaac est la forme suivante :

```
Section Inherit
  + parent_foo:Expanded FOO;
```

L'implantation d'interface donnera lieu à l'utilisation d'héritage alimentaire en Lisaac :

```
Section Insert
  - parent_bar:BAR := BAR;
```

Les classes abstraites du Java seront considérées comme des classes normales. L'absence d'héritage en Java devra se traduire par un héritage à OBJECT en Lisaac du style :

```
Section Inherit
  - parent_object:OBJECT := OBJECT;
```

2.3 Les constructeurs

En java, le constructeur :

```
public foo(int arg)
{
   <instr>
}
```

sera traduit en Lisaac de la manière suivante :

```
- create arg:INTEGER :SELF <-
( + result:SELF;

  result := clone;
  result.make arg;
  result
);

- make arg:INTEGER <-
(
  <instr>
);
```

En Java, l'instanciation :

```
a = new foo(3);
```

sera traduite en Lisaac par :

```
a := FOO.create 3;
```

3 Format du dictionnaire JavaToLisaac

Ce dictionnaire doit être un fichier texte modifiable par n'importe quel éditeur de texte. Le choix d'encodage est laissé à votre appréciation. Néanmoins, réfléchissez sur les points suivants :

- L'ordre des paramètres peut varier ;
- En java, un message est toujours composé d'un seul identifiant et éventuellement plusieurs arguments. En Lisaac, un identifiant de message peut être composé de plusieurs identifiants ayant des arguments entre chacun d'eux.

4 Détection des incohérences

Votre outils doit détecter un certain nombre de cas d'erreur. Ces cas d'erreur doivent être signalés à l'utilisateur à la manière des `warning` dans les compilateurs. Aussi, un commentaire facilement repérable peut être généré dans le fichier de sortie pour repréciser le problème.

- La surcharge d'une méthode n'est pas autorisé en Lisaac ;
- La gestion de `thread` n'est pas encore présente en Lisaac ;
- Certaines briques de base n'ont pas d'équivalent en Lisaac (comme `break`, `continue`, `default`, ...)

5 Simplification ou précision du sujet

- La gestion du dictionnaire n'est pas indispensable ;
- La traduction d'un répertoire complet peut être traité ;

Good luck !