

HMIN210 - Travaux Études et Recherches

Compilation de Common Lisp en Machine Virtuelle Java (JVM)

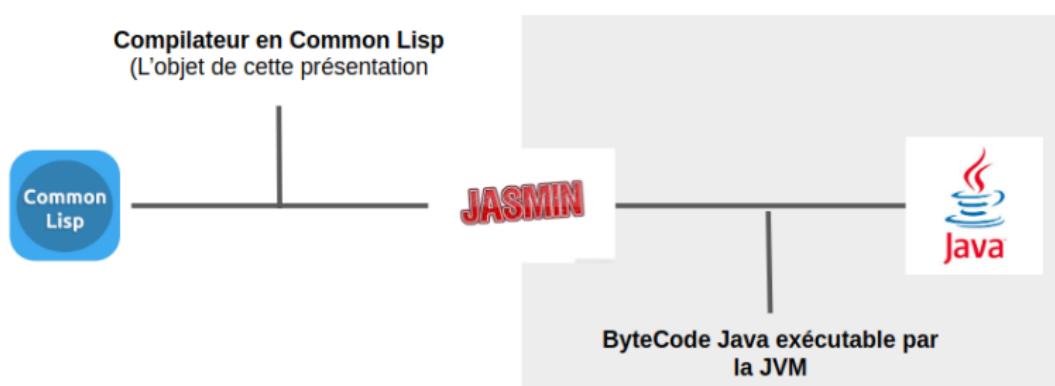
BEYA NTUMBA Joel
MINKO AMOA Imrhan Dareine
ZORO-BI ZAH Jean-Emmanuel

Faculté des Sciences
Université de Montpellier

04 Juin 2019

Résumé

Le but du projet est de mettre en place un compilateur capable de prendre du code source écrit en common Lisp et le traduire en bytecode pour la machine virtuelle java



Contributions

- Bytecode Java humainement illisible
- Plusieurs recherches ont du être effectuées
- Utilité d'utilisation de Jasmin
- Compréhension du fonctionnement de Jasmin...

Sommaire

1 Common Lisp et JAVA

- Syntaxe
- Sémantique
- Grandes différences

2 JVM et Jasmin

- Spécification de la JVM
- Spécification de Jasmin

3 Compilateur Common Lisp vers Jasmin

- Introduction
- Fonctionnalités implémentées du Compilateur
- Extraction de fichiers pour Jasmin

Java

- Blocs de code encadrés par des accolades
- Chaque instruction se termine par un point virgule
- Utilisations d'identifiants permettant de nommer des éléments dans du code source java
- Une variable peut être un type élémentaire soit une classe déclarée sous la forme de classe variable

```
 1 public class Fact {  
 2  
 3     public static int fact (int n) {  
 4         if (n==0) return(1);  
 5         else return(n*fact(n-1));  
 6     }  
 7  
 8     public static void main (String[] args) {  
 9         System.out.print(fact(6));  
10         System.out.print("\n");  
11     }  
12 }
```

Common Lisp

- Notation préfixée parenthésée (`(+ a b)`) au lieu de `a + b`
- Parenthèse ouvrante placée avant la fonction pour les appels de fonction (`(f x)`) au lieu de `f(x)`
- Une expression est soit un atome, soit une liste non vide. Un atome est lui-même un symbole ou un littéral (aussi appelé constante)
- Pas de distinction entre majuscule et minuscule



```
1 (defun factorial (n)
2   "Calcule la factorielle de l'entier n."
3   (if (<= n 1)
4       1
5       (* n (factorial (- n 1)))))

7 (factorial 3)
```

Java

- Java est sensible à la casse (distingue majuscule et minuscule)
- Exemple : "String" n'équivaut pas à "string" en java
- Java structuré en classes
- Chaque classe comprend une ou plusieurs méthodes
- Chaque méthode peut renvoyer un objet ou pas



```
1 public class Fibonacci {  
2     public static long fibonacci(int n) {  
3         if (n <= 1) return n;  
4         else return fibonacci(n-1) + fibonacci(n-2);  
5     }  
6 }
```

Common Lisp

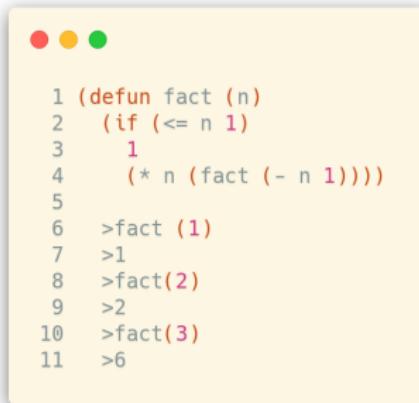
- Sémantique basée sur l'évaluation des expressions évaluables (qui retournent une valeur)
- Toute expression évaluable est une valeur restituée par print et read
- LISP ne fait pas de différence entre opérateurs, mots-clés et fonctions : tout est liste
- list -> (expression)
- expression -> atom | list
- atom -> number | name | string | operator



```
1 (defun fibonacci (n)
2   (if (<= n 1)
3     n
4     (+ (fibonacci (- n 1)) (fibonacci (- n 2))))))
```

Grandes différences

- Common Lisp est un langage impératif et fonctionnel
- Le programme est construit en interaction avec une boucle lecture-évaluation-affichage
- Chaque déclaration est la conséquence d'une interaction.
- Java est un langage orienté objet
- Java est basé sur un modèle édition-compilation-test



```
1 (defun fact (n)
2   (if (<= n 1)
3     1
4     (* n (fact (- n 1)))))
5
6 >fact (1)
7 >1
8 >fact(2)
9 >2
10 >fact(3)
11 >6
```

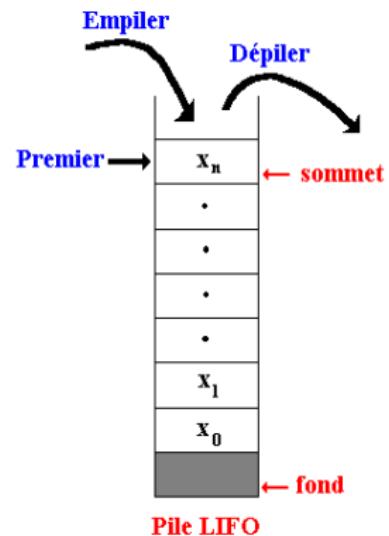
JVM et Jasmin

Spécification de la JVM

Infos sur la JVM

La JVM est basée sur le modèle de la pile
(LIFO, stack-based)

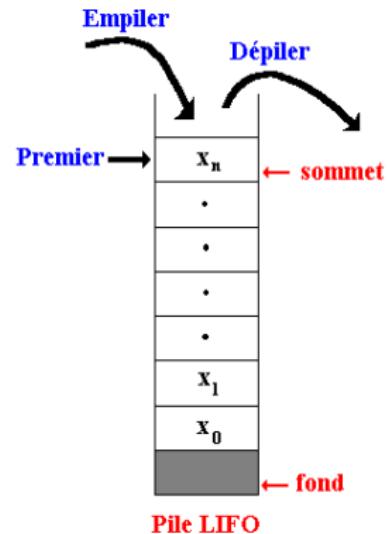
L'instance d'une classe est créée explicitement dans le code et est détruite automatiquement par le ramasse-miette.



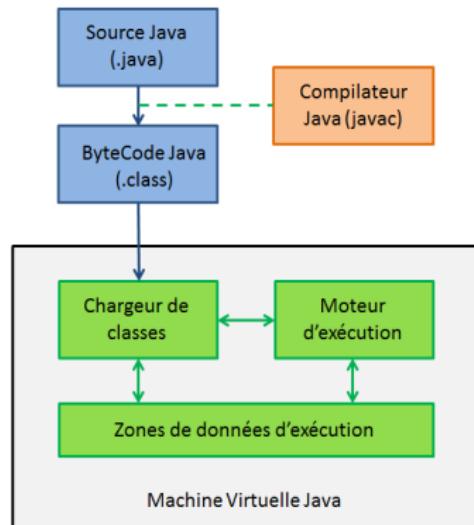
Infos sur la JVM

La JVM est basée sur le modèle de la pile
(LIFO, stack-based)

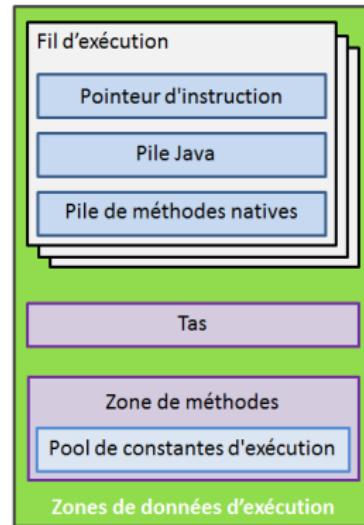
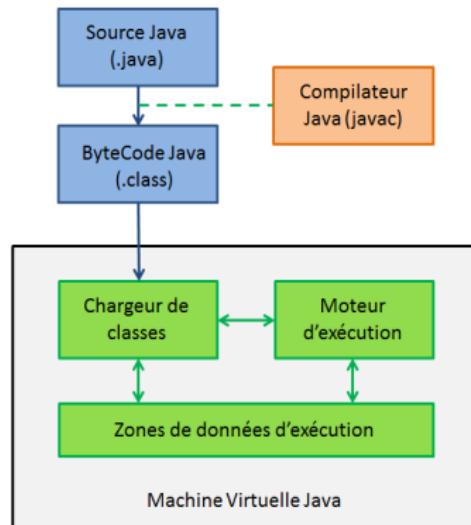
L'instance d'une classe est créée
explicitement dans le code et est détruite
automatiquement par le ramasse-miette.



Zones de données d'exécution



Zones de données d'exécution



Descripteurs

Descripteur	Type
Z	booléen
B	byte
S	short
C	char
I	int
J	long
F	float
D	double
V	void
[<type>	tableau de type <type>
L<type>;	Objet de type <type>

Descripteurs

Signature méthode Java

boolean[] absurde (String s, int i)

Signature méthode bytecode

absurde(Ljava/lang/String;I)[Z

JVM et Jasmin

Spécification de Jasmin

Présentation de Jasmin

- Assembleur pour la machine virtuelle Java
- Projet SourceForge Open Source.
- Écrit par **Jonathan Meyer** en 1996.
- Description de classes Java de manière simplifiée
- Utilise le jeu d'instruction de la JVM
- Converti des fichiers .j en fichiers .class

Présentation de Jasmin

- Assembleur pour la machine virtuelle Java
- Projet SourceForge Open Source.
- Écrit par Jonathan Meyer en 1996.
- Description de classes Java de manière simplifiée
- Utilise le jeu d'instruction de la JVM
- Converti des fichiers .j en fichiers .class

Présentation de Jasmin

- Assembleur pour la machine virtuelle Java
- Projet SourceForge Open Source.
- Écrit par **Jonathan Meyer** en 1996.
- Description de classes Java de manière simplifiée
- Utilise le jeu d'instruction de la JVM
- Converti des fichiers .j en fichiers .class

Présentation de Jasmin

- Assembleur pour la machine virtuelle Java
- Projet SourceForge Open Source.
- Écrit par **Jonathan Meyer** en 1996.
- Description de classes Java de manière simplifiée
- Utilise le jeu d'instruction de la JVM
- Converti des fichiers .j en fichiers .class

Présentation de Jasmin

- Assembleur pour la machine virtuelle Java
- Projet SourceForge Open Source.
- Écrit par **Jonathan Meyer** en 1996.
- Description de classes Java de manière simplifiée
- Utilise le jeu d'instruction de la JVM
- Converti des fichiers .j en fichiers .class

Présentation de Jasmin

- Assembleur pour la machine virtuelle Java
- Projet SourceForge Open Source.
- Écrit par **Jonathan Meyer** en 1996.
- Description de classes Java de manière simplifiée
- Utilise le jeu d'instruction de la JVM
- Converti des fichiers .j en fichiers .class

Factorielle en Jasmin

```
.class public Fact
.super java/lang/Object
.method public static fact(I)I
    .limit locals 1
    .limit stack 3

    iload 0
    ldc 1
    if_icmpgt recursivecall
    ldc 1
    ireturn

recursivecall:
    iload 0
    dup
    ldc 1
    isub
    invokestatic Fact/fact(I)I
    imul
    ireturn
.end method
.end class
```

Compilateur Common Lisp vers Jasmin

Introduction

Common Lisp

Notre programme compile en bytecode Jasmin, mais ce bytecode n'est pas un exécutable à lui tout seul et nécessite la JVM. Il en est de même pour Common Lisp, le programme compilé s'exécute au sein d'un environnement CLisp.

Cependant, Common Lisp demeure un langage à script et Java non. D'où les traitements qui vont suivre.

Introduction

Il a été question de faire le tour de quelques cas d'utilisation incontournables de Common Lisp qu'on a juger important de transformer pour Java.

La prise en compte de la boucle Read-Eval-Print-Loop (REPL), les Lambdas expressions,... Et dans notre cas on s'est plus focalisé sur le REPL.

Prise en compte de l'affichage dynamique des résultats

Nous avons mis directement un point d'entrée (rajout de la fonction main de Java) de toutes les fonctions et méthodes qui seront définies en Common Lisp afin de pouvoir les exécuter directement et afficher les résultats d'exécution.



```
(list ".method public static print(I)V")
(list ".limit locals 1")
(list ".limit stack 2")
(list "getstatic java/lang/System.out Ljava/io/PrintStream;")
(list "iload 0")
(list "invokevirtual java/io/PrintStream/println(I)V")
(list "return")
(list ".end method")
```

Simulation REPL pour Jasmin

En effet, JAVA ne disposant n'en disposant pas. La mise en place de la fonction print du résultat en plus du main rajoutée déjà ci-haut comme complémentaire afin de parfaire une adaptation minime du REPL.

```
(list ".method public static  main([Ljava/lang/String;)V")
(list ".limit locals 1")
(list ".limit stack 3")
(list "aload 0")
(list "iconst_0")
(list "aaload")
(list "invokestatic java/lang/Integer/parseInt(Ljava/lang/String;I)")
(list (concatenate 'string "invokestatic "
  (string-capitalize (list-to-string (list (car exp))))) "/"
  (string-downcase (list-to-string (list (car exp))))) "(I)I"))
(list (concatenate 'string "invokestatic "
  (string-capitalize (list-to-string (list (car exp))))) "/print(I)V"))
(list "return")
(list ".end method")
```

Compilateur Common Lisp vers Jasmin

Fonctionnalités implémentées du Compilateur

Fonction Principale du Compilateur



```
(defun compilation (exp &optional (env ()) (fenv ()) (nomf ()) )
  (let ((arg (if (atom exp) () (cdr exp))))
    (cond
      ((atom exp) (compilation-litt exp env fenv nomf))
      ((member (car exp) '(+ - * /)) (compilation-op exp env fenv nomf))
      ((member (car exp) '(< > = <= >=)) (compilation-comp exp env fenv nomf))
      ((est-cas exp 'and) (compilation-and arg (gensym "finAnd") env fenv nomf))
      ((est-cas exp 'or) (compilation-or arg (gensym "finOr") env fenv nomf))
      ((est-cas exp 'if) (compilation-if arg env fenv nomf))
      ((est-cas exp 'cond) (compilation-cond arg (gensym "fincond") env fenv nomf))
      ((est-cas exp 'progn) (compilation-progn arg env fenv nomf))
      ((est-cas exp 'loop) (compilation-boucle arg env fenv nomf))
      ((est-cas exp 'setf) (compilation-setf arg env fenv nomf))
      ((est-cas exp 'defun) (compilation-defun arg env fenv nomf))
      ((est-cas exp 'let) (compilation-let arg env fenv nomf))
      ((est-cas exp 'labels) (compilation-labels arg env fenv nomf))
      ((and (consp (car exp)) (eql (caar exp) 'lambda)) (compilation-lambda exp env fenv nomf))
      (`(function ,(car exp)) (compilation-appel exp env fenv nomf))))
```

Compilation des opérations arithmétiques



```
(defun compilation-op (exp env fenv nomf)
  (let ((op (car exp)) (arg (cdr exp)))
    (if (null (cddr arg)) (append (compilation (car arg) env fenv nomf)
                                    (compilation (cadr arg) env fenv nomf) (case op
                                      ('+ (list "iadd")) ('- (list "isub"))
                                      ('* (list "imul")) ('/ (list "idiv"))))
        (append (compilation `',(op ,(list op (car arg)
                                             (cadr arg)) ,@(cddr arg)) env fenv nomf))
        )
    )
  )
```

Compilation des structures de condition

```
(defun compilation-if (exp env fenv nomf)
  (let ((sinon (gensym "sinon")) (finSi (gensym "finSi")))
    (append (compilation (car exp) env fenv nomf)
            (compilation (cadr exp) env fenv nomf) (list "ireturn ")
            (list "false:") (compilation (caddr exp) env fenv nomf) (list "ireturn"))))

(defun compilation-cond (exp etiqfin env fenv nomf)
  (if (null exp) () (let ((etiqcond (gensym "etiqcond")))
    (append (compilation (caar exp) env fenv nomf)
            (compilation (cadar exp) env fenv nomf)
            (compilation-cond (cdr exp) etiqfin env fenv nomf)))))
```

Compilation des structures itératives



```
(defun compilation-progn (exp env fenv nomf)
  (if (null exp) () (append (compilation (car exp) env fenv nomf)
    (compilation-progn (cdr exp) env fenv nomf))))  
  
(defun compilation-boucle (exp env fenv nomf)
  (case (car exp) ('while (compilation-while (cdr exp) env fenv nomf))
    ('until (compilation-until (cdr exp) env fenv nomf))))  
  
(defun compilation-while (exp env fenv nomf)
  (let ((fin (gensym "finwhile")) (boucle (gensym "while")))
    (if (eql (cadr exp) 'do) (append (compilation (car exp) env fenv nomf)
      (compilation (caddr exp) env fenv nomf) (error "Syntaxe incorrecte : ~s" exp)))))  
  
(defun compilation-until (exp env fenv nomf)
  (let ((finuntil (gensym "FINUNTIL")) (boucle (gensym "UNTIL")))
    (append (compilation (car exp) env fenv nomf)
      (compilation (caddr exp) env fenv nomf))))
```

Compilation d'opérateurs de comparaison



```
(defun compilation-comp (exp env fenv nomf)
  (let ((op (car exp)) (fin (gensym "finTest")))
    (append (compilation (cadr exp) env fenv nomf)
            (compilation (caddr exp) env fenv nomf) (case op
                  ('= (list "if_icmpeq false")) ('< (list "if_icmpge false"))
                  ('> (list "if_icmple false")) ('<= (list "if_icmpgt false"))
                  ('>= (list "if_icmplt false")))))

(defun compilation-and (exp etiqfin env fenv nomf)
  (if (null exp) () (append (compilation (car exp) env fenv nomf)
                             (compilation-and (cdr exp) etiqfin env fenv nomf))))

(defun compilation-or (exp etiqfin env fenv nomf)
  (if (null exp) () (append (compilation (car exp) env fenv nomf)
                            (compilation-or (cdr exp) etiqfin env fenv nomf))))
```

Compilation de fonctions Common Lisp

```
(defun compilation-defun (exp env fenv nomf)
  (let ((nivem (assoc nomf fenv))) (append (list (concatenate 'string ".class public "
    (string-capitalize (list-to-string (list (car exp)))))) (list ".super java/lang/Object")
    (list (concatenate 'string ".method public static "
      (string-downcase (list-to-string (list (car exp)))))) "(I)I")) (list ".limit locals 1")
    (list ".limit stack 3") (compilation-progn (cddr exp)
      (param-env (cadr exp) env 1 (if nivem (+ 1 (cadr nivem)) 0))
      (fun-env (list exp) fenv (if nivem (+ 1 (cadr nivem)) 0)) (car exp))
    (list "ireturn") (list ".end method") (list ".method public static print(I)V")
    (list ".limit locals 1") (list ".limit stack 2")
    (list "getstatic java/lang/System.out Ljava/io/PrintStream;") (list "iload 0")
    (list "invokevirtual java/io/PrintStream/println(I)V") (list "return")
    (list ".end method") (list ".method public static main([Ljava/lang/String;)V")
    (list ".limit locals 1") (list ".limit stack 3") (list "aload 0") (list "iconst_0")
    (list "aaload") (list "invokestatic java/lang/Integer/parseInt(Ljava/lang/String;)I")
    (list (concatenate 'string "invokestatic "
      (string-capitalize (list-to-string (list (car exp)))))) "/"
      (string-downcase (list-to-string (list (car exp)))))) "(I)I"))
    (list (concatenate 'string "invokestatic "
      (string-capitalize (list-to-string (list (car exp)))))) "/print(I)V"))
    (list "return") (list ".end method")))))
```

Compilation des paramètres



```
(defun compilation-appel (exp env fenv nomf)
  (let ((n (length (cdr exp))) (nivem (assoc (car exp) fenv)))
    (append (compilation-param (cdr exp) env fenv nomf)
            (list (concatenate 'string "invokestatic "
                               (string-capitalize (list-to-string (list (car exp))))) "/"
                               (string-downcase (list-to-string (list (car exp)))) "(I)I")))))

(defun compilation-param (exp env fenv nomf)
  (if (atom exp) () (append (compilation (car exp) env fenv nomf)
                            (compilation-param (cdr exp) env fenv nomf))))
```

Compilation des littéraux



```
(defun compilation-const (exp)
  (list (concatenate 'string "ldc " (write-to-string exp)))))

(defun compilation-varg (exp)
  (list (concatenate 'string "iload 0")))

(defun compilation-litt (exp env fenv nomf)
  (let ((var (assoc exp env)))
    (cond ((not (null var))
            (if (eql (cadr var) 'loc) (if (numberp (cadr var))
                                         (compilation-const (cdr var))))
                ((and (symbolp exp) (not (null exp))) (compilation-varg exp))
                (t (compilation-const exp)))))
```

Extraction dans un fichier .j (Jasmin file)

Nous ré-dirigeons la sortie de l'affichage du code produit vers un fichier .j



```
1 (defun write-to-file (name (compilation code))
2   (with-open-file (stream name :external-format charset:iso-8859-1
3                         :direction :output
4                         :if-exists :overwrite
5                         :if-does-not-exist :create )
6   (format stream content))
7   name)
```

Extension .j : Fichier exécutable en Jasmin

Conclusion

- Ce sujet nous a permis d'en apprendre plus sur le bytecode java
- Aussi de comprendre toutes les instructions qui s'exécutent derrière un compilateur
- Très enrichissant dans le cadre de notre apprentissage

Références

-  Engel, J. *Programming for the JavaTM Virtual Machine.* 1st ed. (Addison-Wesley, 1999).
-  Mak, R. *Writing Compilers and Interpreters: A Modern Software Engineering Approach Using Java®.* 3rd ed. (Wiley Publishing, Inc., 2009).
-  Meyer, J. *JASMIN HOME PAGE.*
<http://jasmin.sourceforge.net/>.
-  Seibel, P. *Practical Common Lisp.* 1st ed. (APress, 2005).

Merci pour votre attention !