Projet Génie Logiciel

Extension [TAB]

Aurélien VILMINOT
Damien CLAUZON
Guilherme KLEIN
Léon ROUSSEL
Pierre ARVY

13 janvier 2022

Table des matières

1 Documentation Java

Deca étant un sous-langage de Java, nous nous intéressons donc aux différentes syntaxes du langage Java. Voici quelques liens qui vont nous être utiles pour étendre la grammaire de Déca donnée dans le polycopié. Dans cette partie, le but n'est pas de construire la grammaire Deca, mais d'étudier et d'extraire des informations sur la syntaxe Java afin de pouvoir ensuite l'adaptée au langage Deca. Des éléments Java qui ne sont pas présents dans le langage Deca seront donc présents dans cette partie, comme les types `short`, `long`,...

1.1 Quelques liens

Spécifications générales
Section 2, grammaires : Decriptions des grammaires
Section 10, tableaux : Description liée au *array*

Syntaxe générale, différente des exemples des sections précédentes, mais plus efficace : Syntaxe

Autre document moins officiel, mais plus explicite

1.2 Description sur les `array` (tableau en français)

1.2.1 Généralités

En Java, les tableaux sont des objets créés dynamiquement. Toutes les méthodes de la classe *Object* peuvent être appelées sur un tableau. Le nombre de composante d'un tableau défini sa taille. Un tableau peut être vide. Tous les éléments d'un tableau sont d'un même type T. Le type du tableau est alors `T[]`. Un tableau peut lui-même contenir des tableaux. Le type des élément d'un tableau est le type des éléments des sous-tableaux les plus profond. Une fois le tableau créé, sa taille ne change pas. La taille d'un tableau peut être trouvé avec l'attribut `length`.

Quelques exemples de déclaration et quelques explications :

- `int[] tab;` ne créé pas les objets du tableau, cela créé uniquement la variable.
- `int[] factorial = { 1, 1, 2, 6, 24, 120, 720, 5040 };` créé effectivement un tableau de `int`, la partie à droite du signe égal est appelée l'initialiseur du tableau.
- `int[] factorial = new int[3];` créé un tableau de `int` initialisé à 0. La partie à droite du signe égal est appelée le créateur de tableau.
- `byte[] rowvector, colvector, matrix[];` est équivalent à `byte rowvector[], colvector[], matrix[][];`
- `float[][] f[][], g[][][], h[];` est équivalent à la liste de déclaration suivante `float[][][][] f; float[][][][][] g; float[][][] h;`

1.2.2 Initialiseur de tableau

L'initialiseur de tableau est une suite d'expressions séparées par des virgules et entouré par des accolades ({}). Une virgule peut apparaître derrière le dernier terme de la suite, elle sera ignorée.

Il doit y avoir une compatibilité entre le type du tableau et le type de la variable assignée. Pour le cas des tableaux, les différents cas suivants doivent être traité comme compatible, et il doit donc y avoir une conversion entre le type de la valeur assignée et le type de la variable qui sera attribuée dans le tableau :

<5.1.1. Identity Conversion>
Toute conversion d'un type vers le même type est permise.

<5.1.2. Widening Primitive Conversion>
Les 19 conversion suivantes :
byte to short, int, long, float, or double

short to int, long, float, or double
char to int, long, float, or double
int to long, float, or double
long to float or double
float to double

<5.1.5. Widening Reference Conversion>
Toute conversion de S vers T si S est un sous-type de T.

<5.1.7. Boxing Conversion>
Type primitif vers l'expression correspondante.
From type boolean to type Boolean
From type byte to type Byte
From type short to type Short
From type char to type Character
From type int to type Integer
From type long to type Long
From type float to type Float
From type double to type Double
From the null type to the null type
<5.1.8. Unboxing Conversion>
From type Boolean to type boolean
From type Byte to type byte
From type Short to type short
From type Character to type char
From type Integer to type int
From type Long to type long
From type Float to type float
From type Double to type double

Si l'expression n'est pas compatible pour être convertie, une erreur est soulevée.

Si l'espace alloué n'est pas assez grand, une erreur de type *OutOfMemoryError* est soulevée, sinon, l'espace est alloué et initialisé avec une valeur par défaut (0, false, null, ... selon le type). Les composantes sont ensuite initialisées avec les valeurs données dans le code source (entre accolades). Si l'initialisation d'une variable échoue, alors l'initialisation du tableau doit échouer. Si la composante est un tableau, alors l'initialisation de la variable doit être effectuer récursivement comme l'initialisation d'un tableau. Cela entraîne donc des déclarations comme celle-ci : `int ia[][] = { {1, 2}, null };`.

1.2.3 Créateur de tableau

Il est également possible de créer un nouveau tableau via une expression de création de tableau. Un exemple de cette création est `int[] factorial = new int[3];`. Tout comme l'initialiseur de tableau, le créateur de tableau doit contenir le type primitif ou la classe des éléments du tableau. La taille du tableau peut être spécifié entre les crochet du créateur de tableau.

1.2.4 Accès à une composante

Pour accéder à une composante d'un tableau, il faut une expression qui référence un tableau suivie d'une expression qui retourne un index entourée de `[ ]`. Par exemple `A[ 2 ]` accède au second élément de A. L'index est une valeur de type int, ou de type short, byte, or char. Il n'est par exemple pas possible d'avoir un index de type long. Les tableaux ont une origine à 0, ce qui signifie que les index possible vont de 0 à n-1 où n est la taille du tableau. La vérification de l'accès à une composante est faite pendant l'exécution, si l'index est strictement négatif ou supérieur strict à n-1, une erreur de type *ArrayIndexOutOfBoundsException* est envoyée.

1.2.5 Assignement

La possibilité d'assignement d'une composante est vérifié pendant l'exécution. Si un tableau est de type A[] et que l'on cherche à modifier une de ses composante, il est vérifié pendant l'exécution que la valeur assignée est de type A. Dans le cas où la valeur assignée n'est pas du même type que A, une erreur de type `ArrayStoreException` est soulevée.

1.2.6 Récapitulatif des méthodes et attributs d'un tableau

Les attributs et méthodes du tableau sont :
- `length`
- La méthode clone qui retourne une copie du tableau principal mais qui partage encore les sous-tableaux.
- Tous les attributs et méthodes de la class Object (sauf `clone`) par héritage des tableaux

Tous les tableaux ont également un object `Class` associé, ce qui permet de tester l'égalité de classe entre différents tableaux.

1.3 Syntaxe

Voici quelques parties de la syntaxe Java qui vont nous être utiles pour la modification des syntaxes Deca.

1.3.1 ArrayCreationExpression (§15.10)

Les règles de grammaires suivantes décrivent la création des tableaux. On a le type qui est écrit par le non-terminal *PrimitiveType*. Puis les crochets avec *Dims* et *DimsExprs* en commençant par les crochet avec des index à l'intérieur.

ArrayCreationExpression :
new PrimitiveType DimExprs Dims$_{opt}$
new ClassOrInterfaceType DimExprs Dims$_{opt}$
new PrimitiveType Dims ArrayInitializer
new ClassOrInterfaceType Dims ArrayInitializer

DimExprs :
DimExpr
DimExprs DimExpr

DimExpr :
[Expression]

Dims :
[ ]
Dims [ ]

1.3.2 ArrayInitializer (§10.6)

Les règles de grammaires suivantes ne servent pas toutes exclusivement pour les tableaux, mais elles aident à comprendre comment arriver au non terminal ArrayInitializer. Pour plus de règles, se référer à la partie 10.6. Array Initializers de la documentation Oracle.

FieldDeclaration :
FieldModifiers$_{opt}$ Type VariableDeclarators ;

VariableDeclarators :
VariableDeclarator
VariableDeclarators , VariableDeclarator

VariableDeclarator :
  VariableDeclaratorId
  VariableDeclaratorId = VariableInitializer

VariableDeclaratorId :
  Identifier
  VariableDeclaratorId [ ]

VariableInitializer :
  Expression
  ArrayInitializer

ArrayInitializer :
  { VariableInitializers$_{opt}$,$_{opt}$ }

VariableInitializers :
  VariableInitializer
  VariableInitializers , VariableInitializer

1.3.3 ArrayAccess (§15.13)

Les règles concernant l'accès à une composante d'un tableau.

ArrayAccess :
  ExpressionName [Expression]
  PrimaryNoNewArray [Expression]

1.3.4 PrimaryExpression

Une des particularités de la syntaxe Java est la restriction de l'accès à un tableau. Un accès à un tableau ne peut pas être fait sur un créateur de tableau. Il y a donc une distinction dans la dérivation de *Primary*. Soit une dérivation vers une création de tableau, soit une dérivation vers un élément qui n'est pas une création de tableau.

Primary :
  PrimaryNoNewArray
  ArrayCreationExpression

PrimaryNoNewArray :
  Literal
  ClassLiteral
  this
  TypeName . this
  (Expression)
  ClassInstanceCreationExpression
  FieldAccess
  ArrayAccess
  MethodInvocation
  MethodReference

2 Choix de l'implémentation en Deca

Dans cette partie, nous spécifions l'implémentation des tableaux en Deca. Cette dernière respectera la syntaxe de Java avec certaines fonctionnalités en moins.

2.1 Généralités

En Deca, les tableaux sont des objets créés dynamiquement. Toutes les méthodes de la classe `Object` peuvent être appelées sur un tableau. Le nombre de composante d'un tableau défini sa taille. Un tableau peut être vide. Tous les éléments d'un tableau sont d'un même type T. Un tableau peut avoir une dimension ou plus, spécifiée par les crochets qui suivent le type. Une fois le tableau créé, la taille de ses dimensions ne change pas. La taille d'un tableau peut être trouvé avec l'attribut `length`. Par exemple `T[][] tab;` est un tableau de type `T[][]`, a deux dimensions, correspond à un tableau de tableaux de type `T[]`, et son attribut length renvoie le nombre de tableaux qu'il contient. Même si un tableau est un objet possédant les méthodes de la classe `Object`, il n'est pas considéré au niveau de la grammaire comme une classe.

Quelques exemples de déclaration et quelques explications :

- `int[] tab;` ne créé pas les objets du tableau, cela crée uniquement la variable.
- `int[] factorial = { 1, 1, 2, 6, 24, 120, 720, 5040 };` créé effectivement un tableau de `int`, la partie à droite du signe égal est appelée l'initialiseur du tableau.
- `int[] factorial = new int[3];` créé un tableau de `int` initialisé à 0. La partie à droite du signe égal est appelée le créateur de tableau.
- `byte[][] matrix, bmatrix;` est équivalent à `byte[][] matrix; byte[][] bmatrix;`
- `float[][] f[][], g[][][], h[];` n'est **PAS** valide. Il faut initialiser les tableaux comme la liste de déclaration suivante : `float[][][][] f; float[][][][][] g; float[][][] h;`

2.1.1 Types supportés

Les tableaux supportent les types `int`, `float` et les classes créés par l'utilisateur.

2.2 Initialiseur de tableau

L'initialiseur de tableau est une suite d'expressions séparées par des virgules et entouré par des accolades ({}). Une virgule peut apparaître derrière le dernier terme de la suite, elle sera ignorée.

Il doit y avoir une compatibilité entre le type du tableau et le type de la variable assignée. Le cas des tableaux est similaire à la compatibilité pour l'affectation existante en Deca.

Si l'espace alloué n'est pas assez grand, une erreur de type *OutOfMemoryError* est soulevée, sinon, l'espace est alloué et initialisé avec une valeur par défaut (0, false, null, ... selon le type). Les composantes sont ensuite initialisées avec les valeurs données dans le code source (entre accolades). Si l'initialisation d'une variable échoue, alors l'initialisation du tableau doit échouer. Si la composante est un tableau, alors l'initialisation de la variable doit être effectuer récursivement comme l'initialisation d'un tableau.

2.3 Créateur de tableau

Il est également possible de créer un nouveau tableau via une expression de création de tableau. Un exemple de cette création est `int[][] factorial = new int[3][5];`. Tout comme l'initialiseur de tableau, le créateur de tableau doit contenir le type primitif ou la classe des éléments du tableau. La taille du tableau spécifié entre les crochet du créateur de tableau doit être une expression retournant un entier. Par exemple, int n; int[] factorial = new int[n = 5]; est valide mais `int[] factorial = new int[5.0];` est invalide. Toutes les dimensions doivent être spécifiées avec ce créateur de tableau. Par exemple, `int[][] factorial = new int[2][];` et `int[] factorial = new int[];` sont invalides.

2.4 Accès à une composante

Pour accéder à une composante d'un tableau, il faut une expression qui référence un tableau suivie d'une expression qui retourne un entier entourée de `[]`. Par exemple `A[2]` accède au second élément de A. Les tableaux ont une origine à 0, ce qui signifie que les entiers possibles vont de 0 à n-1 où n est la taille du tableau. La vérification de l'accès à une composante est faite pendant l'exécution, si l'entier est strictement négatif ou supérieur strict à n-1, une erreur de type *ArrayIndexOutOfBoundsException* est envoyée.

2.5 Assignement

La possibilité d'assignement d'une composante est vérifié pendant l'exécution. Si un tableau est de type A[] et que l'on cherche à modifier une de ses composante, il est vérifié pendant l'exécution que la valeur assignée est de type A. Dans le cas où la valeur assignée n'est pas du même type que A, une erreur de type `ArrayStoreException` est soulevée.

2.6 Récapitulatif des méthodes et attributs d'un tableau

Les attributs et méthodes du tableau sont :

- `length`
- Tous les attributs et méthodes hérités de la class Object (sauf `clone`), c'est-à-dire la méthode `Object.equals`

Tous les tableaux ont également un object `Class` associé, ce qui permet de tester l'égalité de classe entre différents tableaux.

3 Adaptation en Deca

Cette partie est en cours de rédaction, et sera rédigée ultérieurement.