Grenoble INP - Ensimag

Documentation De Conception

PROJET-GL GROUPE 49

BENTAIBI Oussama
AÏT HAMMOU Driss
AJJA Hamza
DAOUD YOUSSEF
LEACHOURI Khalil

2^{nd} Year - 28 janvier 2021

Table des matières

ARCHITECTURE DU COMPILATEUR

I Tree :

La classe **tree** est la plus haute abstraction pour chaque nœud de l'arbre, et donc toutes les autres classes héritent de cette classe.

I.1 AbstractInst :

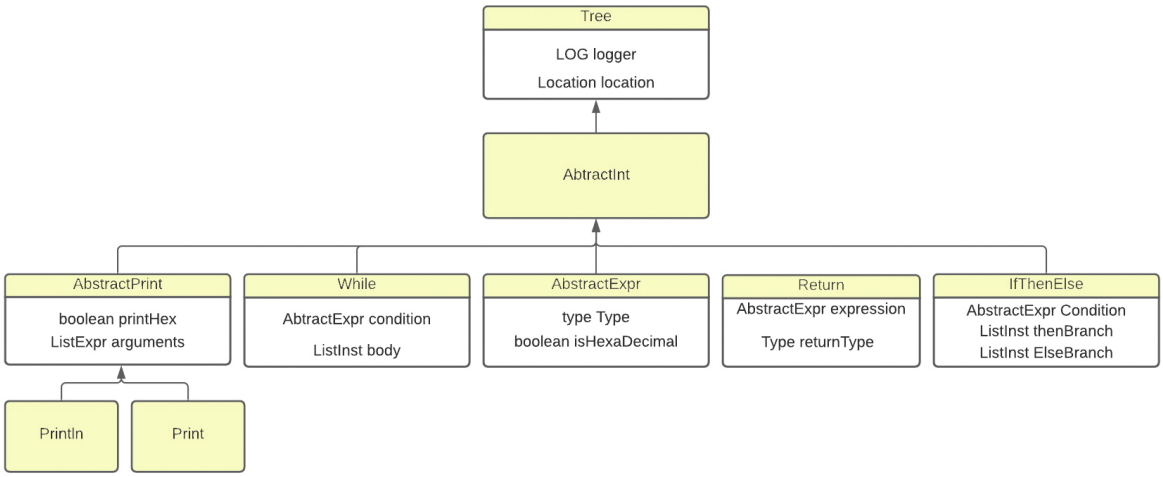

Figure 1: Hiérarchie des classes d'instructions

AbstractInst : Cette classe est une classe abstraite qui fait le traitement des instructions, Elle implémente la vérification contextuelle de l'arbre des instructions ainsi que la génération du code assembleur liés aux instructions de Deca.

I.1.1 AbstractExpr :

AbstractExpr : Cette classe est une classe abstraite qui représente les expressions du langage Deca. Elle possède deux attributs, qui sont ishexaDecimal qui détermine si l'expression est écrite en hexadécimal et le type qui exprime le type de l'expression écrite.

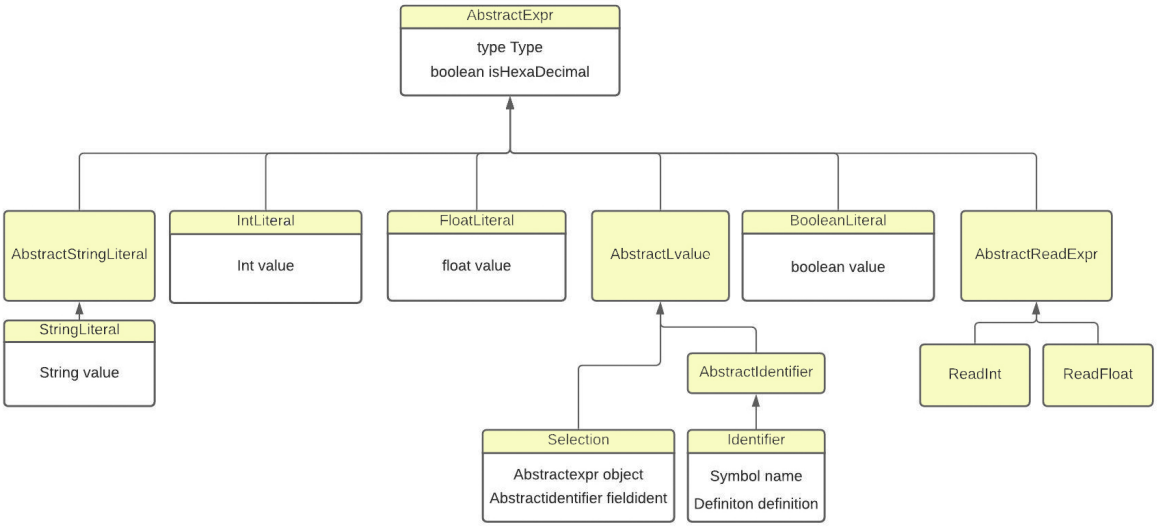

Figure 2: Hiérarchie des classes d'expressions

AbstractLValue : Cette classe correspond à la valeur de la partie gauche d'une affectation. Elle peut être un identificateur ou une sélection de champs.

1-Identifier : Cette classe correspond à un identificateur, c'est-à-dire une chaîne de caractère désignant un objet. Elle possède deux attributs : name qui représente le symbole qui correspond à la chaîne de caractère qui constitue l'identificateur. attributs, name qui représente le symbole qui correspond à la chaîne de caractère qui constitue l'identificateur.

2-Sélection : Cette classe correspond à une opération de sélection de champ dans une classe dans le langage objet. Elle possède deux attributs en particulier : expression qui représente l'expression de la partie gauche de la sélection. un fieldIdent, un identifiant qui représente le champ sélectionné.

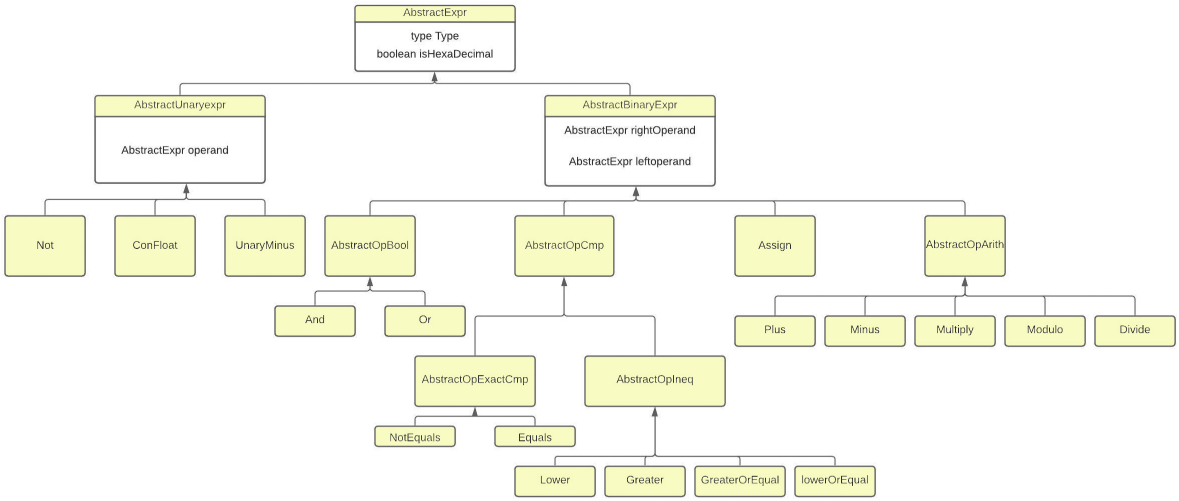

Figure 3: Hiérarchie des opérateurs.

AbstractBinaryExpr : Cette classe représente l'ensemble des expressions qui font intervenir deux opérandes, elle possède donc deux champs correspondant à ces opérandes. Ce sont le leftOperand et le rightOperand.

1-AbstractOpBool : c'est une classe qui hérite de la classe des opérations binaires. Elle représente l'ensemble des opérations booléennes entre deux expressions.

2-AbstractOpCmp : Cette classe correspond à l'ensemble des opérations de comparaison entre deux objets. On distingue deux types de ces opérations. Les opérations de comparaison exactes, c'est-à-dire la différence et l'égalité, et les opérations d'inégalités.

AbstractUnaryExpr : Cette classe représente l'ensemble des expressions qui ne font intervenir qu'un seul opérande. On en a trois, c'est la négation booléenne, l'opposé arithmétique et la conversion implicite d'un entier en un flottant.

AbstractReadExpr : Ce sont les expressions qui permettent la lecture interactive d'un entier ou d'un flottant

StringLiteral : c'est une expression qui représente une chaîne de caractère.

I.1.2 AbstractPrint :

Print : Cette classe est une représentation d'une instruction qui permet l'affichage d'une expression sans le saut de la ligne.

Println : Cette classe est une représentation d'une instruction qui permet l'affichage d'une expression avec le saut de la ligne.

I.1.3 While

While : Cette classe hérite de la classe AbstractInst et correspond à une instruction While. Elle possède deux attributs, condition qui représente la condition de la boucle While. body qui représente la liste des instructions formant le corps de la boucle.

I.1.4 IfThenElse :

IfThenElse : Cette classe hérite de la classe AbstractInst et correspond à une instruction If/Then/else. Elle a trois attributs, condition qui correspond à la condition du if, ThenBranch qui est une liste d'instruction à faire si la condition est vraie, et ElseBranch qui est une liste d'instructions formant le corps de la branche Else.

I.2 AbstractProgramm :

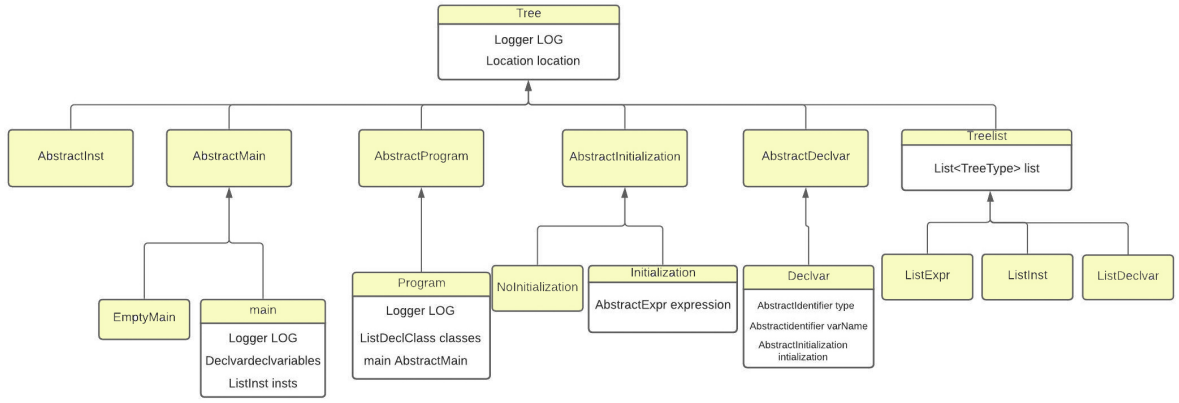

Figure 4: Hiérarchie des classes (Language sans objet)

AbstractProgramm : Cette classe représente tout le programme Deca, elle possède deux attribut : main qui représente le programme principal et classes qui représente les classes du programme.Cette classe représente tout le programme Deca, elle possède deux attribut : main qui représente le programme principal et classes qui représente les classes du programme.

I.3 AbstractMain :

Main : Cette classe correspond au programme principal, elle possède deux attributs : declVariables qui représente les variables déclarées pour le main, et insts qui représente la liste des instructions pour le programme.

EmptyMain : Cette classe correspond à un programme vide.

I.4 AbstractDecVar :

DeclVar : Cette classe correspond à la déclaration d'une méthode dans le main ou bien au sein d'une méthode. Elle possède trois attributs : varName qui représente le nom de la variable, type qui correspond au type de la variable et initialization qui correspond à l'initialisation de la variable.

I.5 AbstractInitialization :

Initialization : cette classe correspond à l'initialisation d'une variable ou d'un champ. elle possède comme attribut une expression 'expression' qui doit être affecté à la variable ou au champ.

NoInitialization : cette classe sert lorsque la variable ou le champ n'est pas initialisé.

I.6 AbstractDecClass :

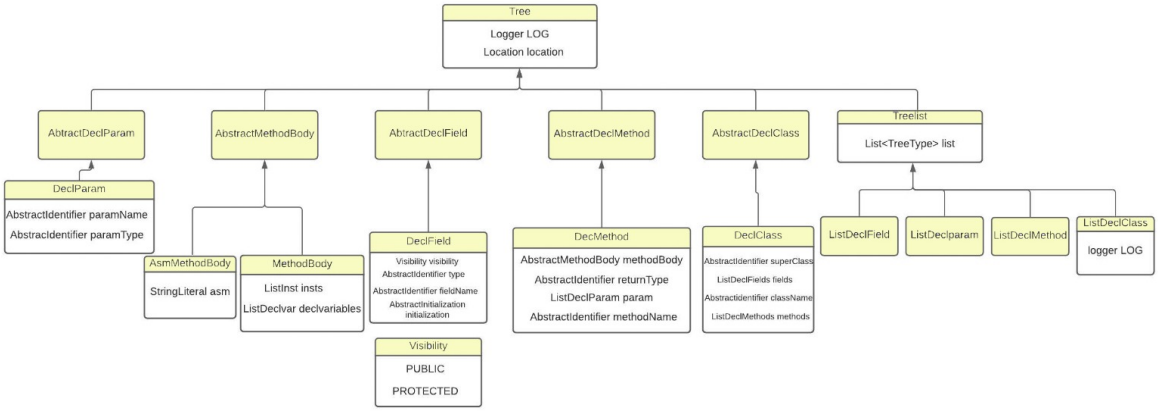

Figure 5 : Hiérarchie des classes (partie objet)

DeclClass : Cette classe correspond à la déclaration d'une classe, elle sert pendant les trois passes de la vérification contextuelle. Elle possède quatre attributs : classname, un identifiant qui représente le nom de la classe, superClass, un identifiant qui représente le nom de la classe mère. fields et methods qui représente deux listes de déclaration de champs et de méthodes respectivement.

I.7 AbstractDecField :

DeclField : Cette classe correspond à la déclaration d'un champ, elle a trois attributs. Elle permet cela de la même façon que la déclaration d'une variable sauf qu'on a ici un attribut supplémentaire "visibility" qui permet de savoir si le champ est protégé ou publique.

I.8 AbstractDecMethod :

DeclMehtod : C'est une classe qui permet la déclaration d'une méthode. Elle a quatre attributs : returnType qui permet de déterminer le type de retour de la méthode, methodename, un identifiant qui présente le nom de la méthode, param qui représente les paramètres de la méthode et methodBody qui représente le corps de la méthode.

I.9 MethodBody :

MethodBody : correspond au corps d'un méthode, c'est-à-dire un corps qui ressemble à un programme principal.

I.10 MethodAsmBody :

methodAsmBody : Correspond au corps d'une méthode assembleur. Il est constitué d'un StringLiteral asm qui correspond à la chaîne de caractères de l'assembleur devant être effectué par la méthode.

I.11 Autre Class utiles :

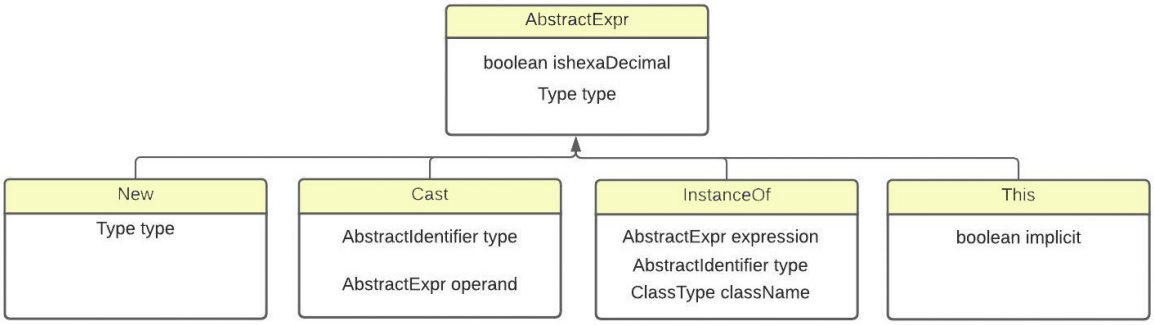

Figure6: Hiérarchie de quelques expressions supplémentaires

Declparam : Cette classe correspond à la déclaration d'un paramètre. Elle possède deux attributs : type qui correpond au type de paramètre et name qui est un identifiant correspondant au nom du paramètre.

New : Cette classe correspond à la création d'un nouvel objet de type classe.

Cast : Cette classe est une expression qui permet de faire une conversion. Elle possède deux attributs. une expression operand qu'il faut convertir, et type qui représente le type auquel on va convertir cette expression.

instanceOf : C'est une expression qui permet de déterminer si un objet appartient à une classe. This : C'est une classe qui permet de faire référence à la classe courante.

MethodCall :c'est une classe qui permet de faire un appel à une méthode.

II Contexte :

II.1 Définition :

Definition : Cette classe correspond à la définition associée à un identificateur dans un environnement de types ou d'expressions. Elle a deux attributs : location qui correspond à l'emplacement de l'identificateur, et type qui représente le type de l'identifiant auquel la définition est associée.

II.1.1 Type Definition

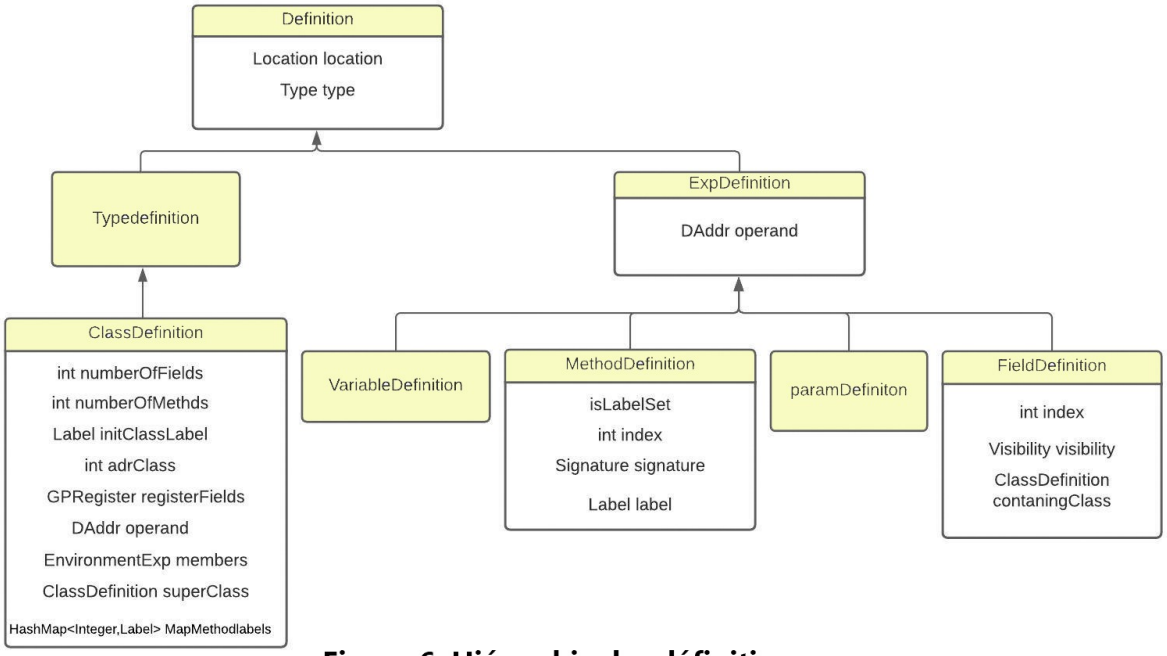

Figure 6: Hiérarchie des définitions

TypeDefinition : Cette classe hérite de la définition et permet de savoir la définition d'un type associée à un identificateur dans l'environnement type.

ClassDefinition : C'est un classe TypeDéfinition particulière aux classes.

II.1.2 Exp Definition

ExpDefinition : Elle permet de donner une définition d'une expression associée à un identificateur dans l'enviremment des expressions.

variableDefinition : C'est une ExpDefinition particulière aux variables.

Methoddefinition : C'est la classe ExpDefinition particulière aux méthodes.

FieldsDefinition : C'est la classe ExpDefinition particulière aux champs.

ParamDefinition : C'est la classe ExpDefinition particulière aux paramètres de méthodes.

II.2 Types :

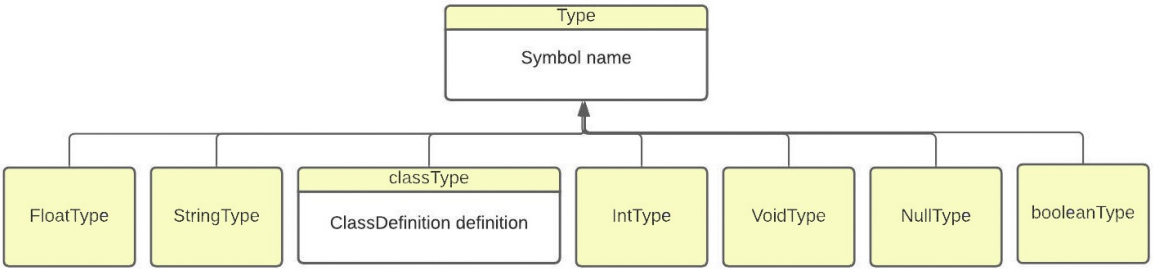

Figure 8: Hiérarchie des types

Type : Cette classe représente les différents types que peuvent prendre les identificateurs dans un programme.

II.2.1 FloatType :

FloatType : Classe particulière au type flottant.

II.2.2 IntType :

IntType : Classe particulière au type entier.

II.2.3 BooleanType :

BooleanType : Classe particulière au type boolean.

II.2.4 StringType :

StringType : Classe particulière au type des chaines de caractères.

II.2.5 VoidType :

VoidType : Classe particulière au type void.

II.2.6 NullType :

NullType : Classe particulière au type null.

II.2.7 ClassType :

ClassType : Classe particulière au type class. elle admet un attribut particulier au type class ; definition qui correspond à la définition d'une classe.

III codegen :

Ce package contient les classes nécessaire à la génération du code assembleur :

III.1 GestionMemoire :

Cette classe permet la gestion de mémoire. Il stocke le dernier registre utilisé, il donne un nouveau registre, ainsi, lors de l'utilisation de tous les registres, il permet une allocation dans la pile à travers des PUSH et POP.

III.2 GestionException :

Cette classe contient la méthode "addExceptions()" qui permet d'ajouter les labels de toutes les exceptions possible à la fin du fichier ".ass" pour pouvoir les récupérer en cas de besoin sans se soucier de leur présence.

III.3 GestionLabel :

Cette classe permet de gérer les différents labels. Ainsi, en donnant à chaque label un nom lié à un nombre qu'on incrémente à chaque fois dans cette classe, on peut dire qu'elle sert aussi à associer à chaque label un nom unique afin d'éviter les erreurs liés au nommage.

IMPLÉMENTATION DU COMPILATEUR :

IV arbre de compilation :

La création de l'arbre commence tout d'abord par la reconnaissance des tokens du langage Deca. Pour cela, il faudra compléter le fichier **Decalexer.g4**. On illustrera cette création par un exemple de la création d'un nœud de print. On ajoutera donc le mot 'print' dans notre fichier lexer. La création de l'arbre se fera lors de la seconde vérification syntaxique. Il faudra donc écrire un class Java correspondant (une classe **Print.java** par exemple). Afin de respecter les notions d'héritage, il faudra que cette classe hérite de la classe AbstractInst, puisqu'un Print est une instruction (l'instruction Print étend une classe AbstractPrint qui étend à son tour **AbstractInst**. La création du nœud correspondant dans l'arbre se fait en instanciant un objet du type adéquat dans la règle associée dans le fichier **DecaParser.g4**. Il faut donc créer une règle décrivant la syntaxe utilisée pour la nouvelle fonctionnalité et ajouter le code nécessaire pour instancier l'objet et lui affecter sa location dans le programme. C'est le travail que nous avons effectué : On a ajouté toutes les classes nécessaires pour le langage objet, tandis que les classes nécessaires pour le langage objet étaient déjà présentes.

V Parcours de l'arbre :

Le parcours de l'arbre permet de faire deux fonctionnalités : vérifier si le programme est valide contextuellement et décorer cet arbre en ajoutant les types et les définitions nécessaires.

Les trois passes de cette vérification se font en implémentant des méthodes **verifyXxxx**. Cela permet de parcourir l'arbre en itérant sur les enfants de chaque nœud. Prenons encore l'exemple du **Print.** Pour que cette instruction soit correcte contextuellement, il faut vérifier que les types des expressions passées en argument doivent être des types imprimables avec un **verifyExpr** de la classe **AbstractExpr** qui retourne le type de l'expression. C'est-à-dire qu'ils soient entiers, flottants ou chaînes de caractères. Il faut en plus faire une set sur les types de ces expressions.

VI Méthodes particulières de vérifications :

Construction de la signature d'une méthode : Lors du deuxième passe, nous avons choisi de créer une méthode **createSignature()** de le fichier **ListDelcParam**. Il s'agit de parcourir la liste des paramètres, vérifier leur type via la méthode **verifyType** que nous avons créé dans **Declparam**, puis ajouter leurs signatures dans une liste de signatures vide qu'on a déjà créé.

VII Vérification séparée des initialisations :

Pour permettre de déclarer les champs dans n'importe quel ordre donné par l'utilisateur. On a séparé la vérification des initialisations des champs en créant une fonction **verifyDeclListFieldInit** qui s'occupe de vérifier la validité des initialisations séparément à la fonction **verifyListDeclField** qui s'occupe de vérifier les visibilités, les noms et le types des champs.