Equipe GL33

Documentation de Conception

DecacCompiler:
DecacCompiler s'occupe de stocker la liste des symboles dans l'attribut symbols. Cela permet de comparer les différents symboles rapidement en faisant des comparaisons de pointeurs. La fonction getSymbol crée un nouveau symbole ou retourne un symbole en fonction de si le symbole existait déjà.

DecacCompiler s'occupe de stocker la liste des type env_types dans l'attribut envType. envType est remplie par défaut par les types void, boolean, float, int, String, null, Object.

DecacCompiler s'occupe de gérer les registres. Les registres sont utilisés dans l'ordre et s' il n'y a plus de registre vide, le registre max est PUSH sur le stack. Il faut toujours utiliser popReg après avoir utilisé getNewReg après avoir fait les calculs.
De plus, **DecacCompiler** compte de nombre max de registres utilisés ce qui permet de savoir combien de registre il faut push dans les méthodes et quelle place il faut pour pouvoir PUSH les registres.

DecacCompiler stock aussi le nombre max de paramètres des fonction du bloc.

DecacCompiler stock le label de fin de méthode.

ClassDefinition:
ClasseDefinition stock en plus le label correspondant à la méthode d'init de la classe et l'adresse de la table des méthode de la classe.

EnvironementExp:
Une HashMap est utilisée pour stocker les définitions et pour pouvoir les récupérer rapidement.
Une méthode getMethods parcourt toutes les définitions pour récupérer toutes les méthodes de la classe courante et les retourne dans l'ordre de leur index.

Classes dans Tree:
AbstractBody: correspond à un bloc, deux sous classes **BlockBody** et **AsmBody**.
BlockBody correspond aux blocs de code deca et gère la vérification de stackoverflow, de return et push les registres.
AsmBody correspond à du code en assembleur. Il ne peut être utilisé que dans les méthodes.

Cast correspond au cast entre différentes classes.
Attribute correspond à l'accès au attribut d'une classe (ex: obj.attr)

AbstractDeclField et **DeclField** correspondent à la déclaration d'attribut.
AbstractDelcMethod et **DeclMethod** correspondent à la déclaration de méthode.

FunctionCall et **MethodCall** servent à l'appel de méthode.
FunctionCall : function(args..)
MethodCall : obj.method(args…)
FunctionCall n'est pas utilisable dans main.

ListParameter et **Parameter** est utilisé dans la déclaration de méthode pour stocker le nom et type de chaque paramètres.