Le but de ce TP est de programmer un contrôle de type pour Ava, basé sur le parcours par un visiteur de l'arbre abstrait du TP4. Le contrôle de type doit :

- remplir la table des symboles ;
- lever une exception de type `TypeCheckingException` en cas d'erreur de type, double déclaration, variable non déclarée, etc.

Le support de cours présente de nombreux exemples pour vous aider. Il est impératif d'avoir terminé le TP4 avant d'attaquer ce TP.

1 Description du matériel fourni

Copiez chez vous l'archive `tp5.tgz` qui est sur le portail. Elle contient des scripts, des sources, et la structure de répertoire à laquelle vous êtes maintenant habitués.

Sources

Paquetage `ava.tableSymboles` Le répertoire `src` contient tous les sources fournis jusquà présent, avec en plus un paquetage `tableSymboles` qui, comme son nom l'indique, contient ce qu'il faut pour créer et remplir une table des symboles pour Ava :

- `AttributsIdentificateur` contient les attributs d'une variable, ici on utilisera son type ;
- `TableSymboles` permet d'associer à un `String` un `AttributsIdentificateur` ;
- `Type` contient l'ensemble des types Ava sous forme d'un type énuméré.

Pour plus de détails, consulter la doc (`ant genDoc`).

Paquetage `ava.typeChecking` Une ébauche de paquetage `typeChecking` est aussi fournie, sous la forme d'un squelette de classe `typeChecking.TypeChecker` et d'une classe `typeChecking.TypeCheckingException`.

Scripts

Les scripts habituels sont fournis, avec de nouveaux venus :

- `execEnLigneTypeChecker.sh` lance le contrôleur de type sur l'arbre abstrait d'un programme entré dans la console, et lève une `TypeCheckingException` en cas d'erreur de type ;
- `execSurFichierTypeChecker.sh` qui prend en ligne de commande le nom d'un fichier à analyser, et lance le contrôleur de type sur l'arbre abstrait correspondant au programme contenu dans le fichier, avec le même comportement concernant les erreurs ;
- `execTestTypeChecker` lance le contrôle de type sur tous les tests des répertoires `test/OK` (programmes sémantiquement corrects) et `test/KO` (programmes sémantiquement incorrects).

2 Travail à réaliser

Avant toute chose, commencer par relire la spécification de Ava pour identifier quels sont les contrôles à effectuer.

Ensuite copier les spécifications du TP4 dans le répertoire `spec`, puis générer les analyseurs par `ant genAnLex` et `ant genAnSynt`.

Votre travail est maintenant de compléter la classe `typeChecking.TypeChecker`. Le squelette fourni contient juste de quoi assurer la compilation de la classe principale `LanceurTypeChecker` : l'exécution affiche pour n'importe quel programme Ava une table des symboles vide.

Commencer par ajouter ce qu'il faut pour que `TypeChecker` soit un `Visiteur`. Ensuite compléter le corps des méthodes de manière itérative et incrémentale : coder un petit peu, compiler, tester en remplissant `test/OK` et `test/KO` au fur et à mesure, en s'aidant de `execEnLigneTypeChecker` et `execSurFichierTypeChecker` pour la mise au point.