TP1 : Analyseur lexical

1- Déclarer les tokens sous forme d'un ensemble énuméré : TOKENS.

- Créer un **nouveau projet Java** dont le nom est « **mips1Proj** »
- Créer un **nouveau package** dont le nom est « **net.mips.compiler** »
- Créer un **nouveau ENum** dont le nom est « **Tokens** » et déclarer dedans les codes de symboles **ID_TOKEN**, **NUM_TOKEN**, … comme indiqué dans le transparent 7.

2- Déclarer le type des erreurs sous forme d'un ensemble énuméré : CODES_ERR.

- Créer un **nouveau ENum** dont le nom est « **CodesErr** » et déclarer dedans les codes d'erreurs lexicales **CAR_INC_ERR**, **FIC_VID_ERR** comme indiqué dans le transparent 13.

3- Déclarer la table des erreurs (code erreur, message associé).

- En Java, on n'a pas besoin de définir un tel tableau. Puisque son rôle est de faire l'association entre le code de l'erreur et le message d'erreur, on va modifier la classe « **CodesErr** » de la manière suivante :
    - Passer à chaque constante un message d'erreur : CAR_INC_ERR(″Symbole inconnu″) et FIC_VID_ERR(″Erreur d'ouverture de fichier″).
    - Déclarer un attribut **private message** de type **String**.
    - Définir un constructeur **private** pour initialiser cet attribut.
    - Générer les getters et setters de l'attribut message.

4- Déclarer le type symbole.

- Créer une nouvelle classe « **Symboles** » dans laquelle vous déclarer les attributs token de type Tokens et nom de type String. Générer le code du constructeur et les getters et setters de chaque attribut (voir pour cela le transparent 8).

5- Déclarer la table des mots clés.

- Il s'agit d'une table qui sera utilisée par l'analyseur lexical. Par conséquent, on doit définir une nouvelle classe « Scanner » dans laquelle on va implémenter tout ce qui reste :
    - Déclarer dans cette classe l'attribut **motsCles** de type **List<Symboles>** ou **ArrayList<Symboles>**. List<T> et ArrayList<T> sont définis dans le package java.io.

6- Déclarer les variables globales (symb_cour, car_cour, fich_sour).

- Dans la classe Scanner, déclarer les attributs :
    - symbCour de type Symboles.
    - carCour de type Char.
    - fluxSour de type FileReader.
- FileReader du package java.io est une classe permettant de manipuler les fichiers texte en mode lecture. La création d'un flux consiste à instancier la classe FileReader en transmettant à son constructeur un objet de type File du package java.io. Avec un objet de type File associé à un nom de fichier, on peut vérifier si ce fichier existe en appelant la méthode exists(). Avec l'objet de type FileReader, on peut lire un caractère en appelant la méthode read(). Mais elle doit être précédée par un test vérifiant s'il y a encore des données à lire, on appelle pour cela la méthode ready() qui retourne un booléen. Dans

leurs exécution, ces deux méthodes peuvent lever une exception de type IOException, d'où il est nécessaire de faire un throws dans l'entête de la méthode qui les appelle.

- Déclarer un constante EOF.
- Définir un constructeur auquel on transmet le nom du fichier à analyser. Dans le constructeur, créer un objet File, tester si le fichier existe et sinon lever une exception de type ErreurLexicale. Initialiser l'attribut fluxSour avec une nouvelle instance de la classe FileReader et initialiser l'attribut motsCles avec une nouvelle instance de la classe ArrayList.
- Générer les getters et setters pour chaque attribut.

7- Ecrire une procédure qui initialise la table des mots clés.

- Dans la classe Scanner, définir la méthode initMotsCles() en y insérant les mots clés sous forme d'objets de type Scanner.

8- Ecrire la procédure ERREUR(code d'erreur) qui affiche le message d'erreur associé et stoppe l'exécution.

- En java, on n'a pas besoin de définir la procédure erreur() mais on doit gérer les exceptions. Pour cela :
  - On va définir une nouvelle classe ErreurCompilation qui hérite de la classe prédéfinie Exception en y définissant un constructeur avec un argument de type String représentant un message d'erreur de compilation.
  - Ensuite on définit la classe ErreurLexicale qui hérite de la classe ErreurCompilation en y définissant un constructeur avec un argument de type CodesErr.
- En cas d'erreur lexicale, on lance une exception avec l'instruction throw et en créant un objet de type ErreurLexicale. Une méthode qui lors de son exécution peut lancer une exception doit le déclarer dans son entête à l'aide de throws.

9- Ecrire la procédure Codage_Lex qui reçoit en entrée la valeur d'un mot et retourne son token.

- Dans la classe Scanner, définir la méthode codageLex() qui permet de déterminer le token d'un symbole mot clé ou identificateur.

10- Ecrire la procédure LIRE_CAR(), LIRE_MOT(), LIRE_NOMBRE(), SYMB_SUIV(), etc.

- Dans la classe Scanner, définir les méthodes :
  - lireCar() qui lit un caractère s'il existe ou lit un EOF.
  - lireNombre() et lireMot() qui doivent être implémentées comme expliqué dans le transparent 11.
  - symbSuiv() qui doit être implémentée comme expliqué dans le transparent 12.

11- Tester votre analyseur lexical

- Pour tester l'analyseur, on édite le programme du transparent 14. Après on définit la méthode main() dans la classe Scanner. Dans le main(), on crée un objet de type Scanner en précisant le fichier contenant le programme à analyser, puis on initialise la table des mots clés, ensuite on lit le premier caractère. Finalement tant qu'on a pas atteint le EOF, on appelle de manière itérative symbSuiv() et on affiche le token du symbole courant.