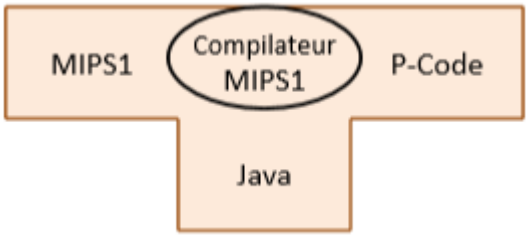
Compte rendu Compilation :

Premier partie :

1) Objectif :

L'objectif de ces premier TP c'est de réaliser un compilateur avec le langage java qui génère du P-code et ceci est à partir d'un programme mini Pascal.



2) L'analyseur lexical :

L'analyseur lexical (Tokenizer) est le processus qui transforme une chaine de caractère à un ensemble de tokens et ceci en se basant sur des règles lexical: une ou plusieurs expressions régulières qui définissent la structure lexical du langage (les mots clés et les identificateurs). Le tokenizer ne prend pas la charge de vérifier la grammaire ou la sémantique du langage.

Il y a deux façons pour créer un Analyseur lexical, soit en développant à la main (c'est le cas dans notre TP), soit en utilisant des outils qui le génèrent automatiquement (Flex/lex, JFlex …)

Exemple d'une règle lexical :

ID ::= lettre{lettre | chiffre} *

a) Implémentation de l'analyseur lexical :

Le codage de tokenizer est très simple, on se basera principalement sur deux classes et une énumération.

Dans la classe **Token** de type **enum,** on va implémenter tous les tokens des symboles possibles de notre mini-Pascal.

La classe **Symbole**, elle définit l'objet symbole qui se compose principalement d'une chaine de caractère –nom– (la forme textuel du symbole) associé à un objet de type Token –token–.

Finalement la classe **Scanner,** comme son nom l'indique elle est chargée de scanner le fichier d'entré mot par mot et dégagée les tokens si trouvés en sortie sinon déclencher une erreur si le **nom** d'un symbole ne matche aucun token (ne respect aucune règle lexical).

L'objet scanner est composé de quatre variable principale :

- carCour : le dernier caractère lu;
- symbCour : le symbole lu qui un ou plusieurs carCour ;
- fluxSour : l'objet responsable des actions d'entré (Lecture du fichier)
- motscles : un tableau contenant tous les mots clés du langage.

La procédure de l'objet **Scanner**, est de lire le fichier d'entré caractère par caractère via les méthodes **lireCar(), lireMot() ou lireNombre()** implémentées dans la procédure **symbSuiv()**, à chaque fois qu'on lit un symbole on cherche dans la table motscles, si ce dernier existe on lui associe son token sinon on le considère comme étant un identificateur token.

N.B : Pour plus de détail veuillez voir le code source des classes Scanner, Symbole et Tokens u package **net.mips.compiler**.

3) L'analyseur Syntaxique :

Son rôle est de vérifier si les Tokens générés forment une expression significative. Cela fait appel à une grammaire sans contexte qui définit des procédures algorithmiques pour les composants. Ces procédures fonctionnent pour former une expression et définir l'ordre particulier dans lequel les Tokens doivent être placés.

Le principe de l'analyseur syntaxique (parser), on associe pour chaque règle syntaxique une procédure de teste qui vérifie qu'on a le token attendu si oui on passe au symbole suivant, sinon on lance une erreur syntaxique.

Le parser prend en entré un flux de tokens et génère en sortie une arbre syntaxique.

```
Procedure TestAccept( t : Token, c : CodeERR)
Debut
    Si (symbCour.token  =  t) alors
            SymbSuiv()
    Sinon
            Erreur(c)
    Fin si
Fin
```

a) Exemple de règle syntaxique :

PROGRAMME ::= **program ID ; BLOCK .**

Comme on peut voir cette règle syntaxique représente un ensemble de procédures de testes qui définit la grammaire **hors contexte** que doit suivre notre programme en mini Pascal.

b) Codage du Parser :

Alors pour la réalisation de notre analyseur syntaxique on aura besoin principalement de trois classes :

- **CodesErr :** une énumération dans laquelle on associe pour chaque token un message d'erreur.
- **ErreurSyntaxique :** une classe qui hérite bien évidement de ErreurCompilation est de type Exeption, elle prend en paramètre un objet de type CodesErr.
- **Parser :** c'est la classe qui définit toute les procédures des règles syntaxiques.

N.B : Pour plus de détail veuillez lire les commentaires dans le code intégré dans chaque classe.

4) L'Analyse Sémantique :

La sémantique aide à interpréter les symboles, leurs types et leurs relations les uns avec les autres. L'analyse sémantique juge si la structure syntaxique construite dans le programme source en tire un sens ou non.

Dans notre mini Pascal les identificateurs sont les obejt sémantique du programme.

a) Les règles sémantiques :

- **Règle 1 :** toutes déclarations dans CONST et VARS.
- **Règle 2 :** pas de double déclaration.
- **Règle 3 :** Après begin tous les symboles doivent être déjà déclarés.
- **Règle 4 :** une constante ne peut changer de valeur dans le programme.
- **Règle 5 :** L'ID du programme ne peut pas être utilisé dans le programme.