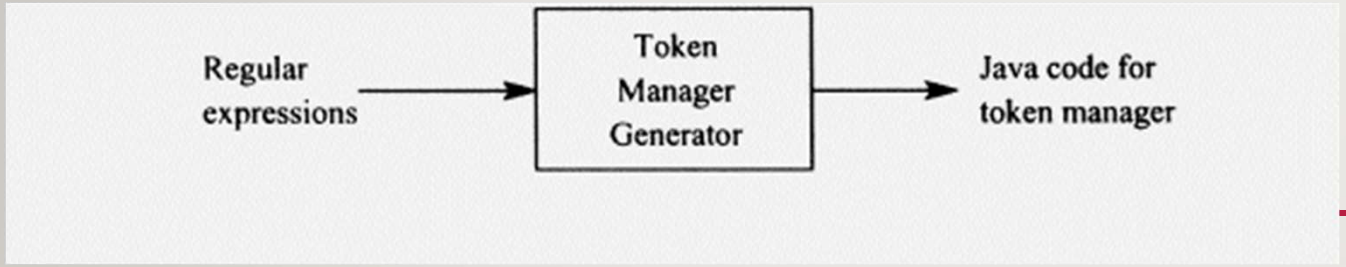
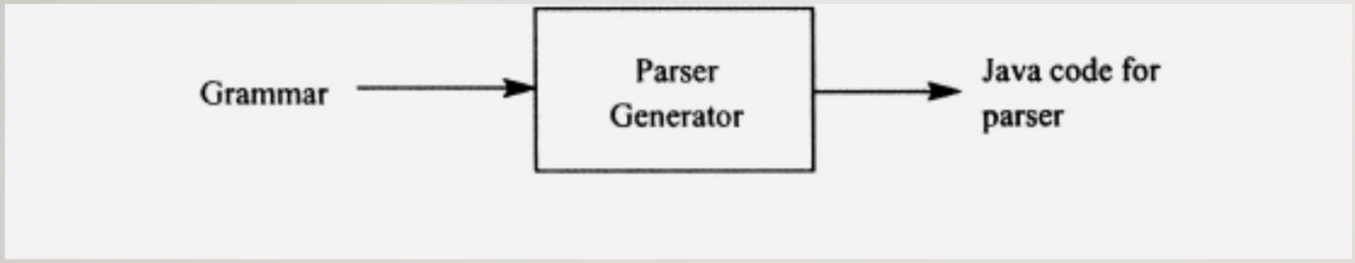
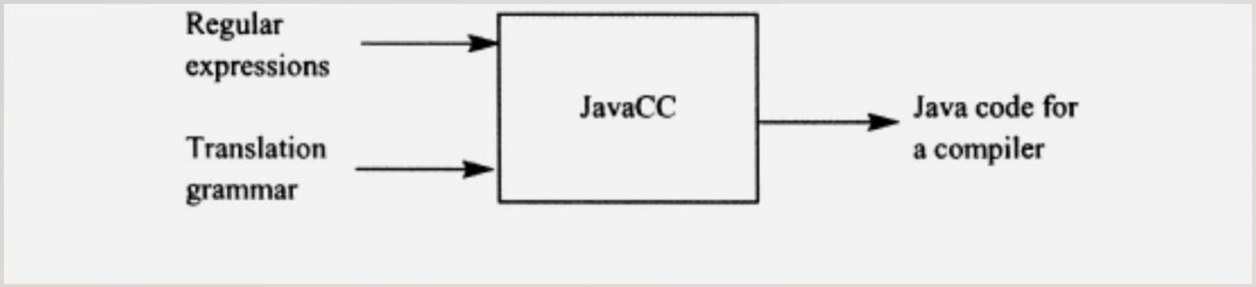
JAVACC: PRISE EN MAIN

JAVACC

JAVACC (JAVA COMPILER COMPILER)

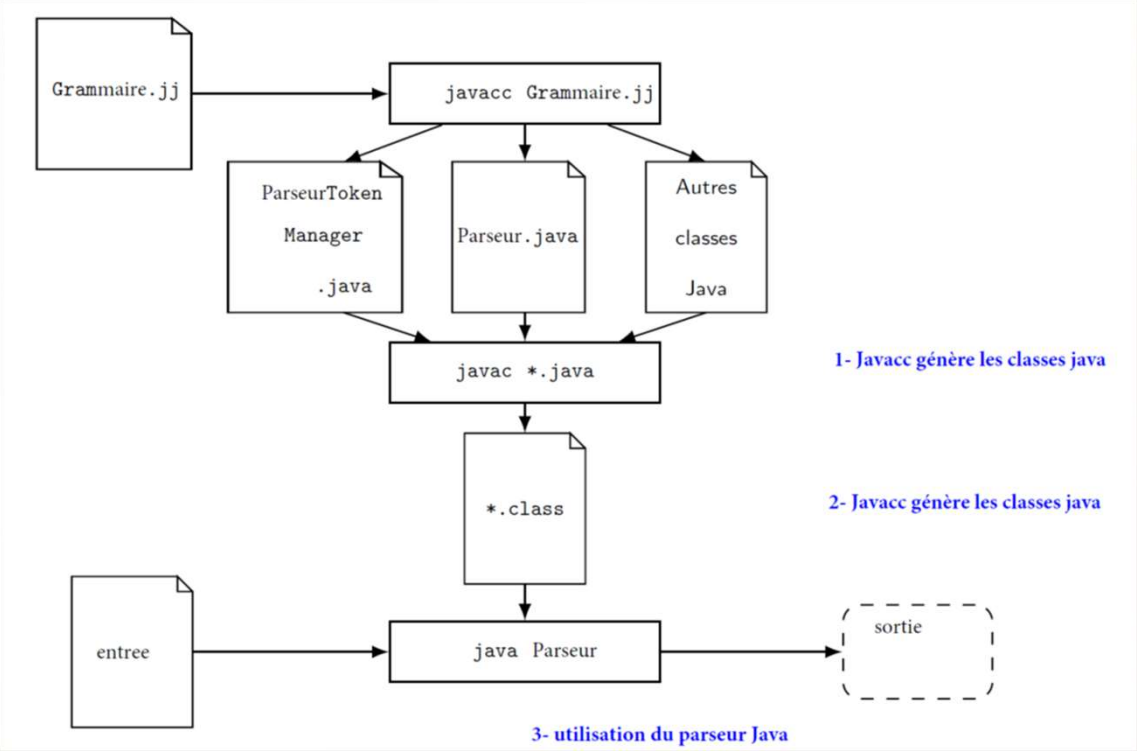






- Génère des parseurs TOP-DOWN
- Génère des parseurs récursifs descendants LL(K), écrit une méthode pour chaque non terminal
- http://javacc.org/

FONCTIONNEMENT JAVACC



- Token Manager: reçoit en entrée une séquence de caractères et retourne une séquence de tokens
- Parseur: analyse la séquence des tokens

STRUCTURE DU FICHIER GRAMMAIRE.JJ

Le fichier de description de la grammaire peut être composé des sections suivantes:

1. options (optionnel)
2. Code java entre les éléments PARSER_BEGIN et PARSER_END
3. Une section optionnelle pour les déclarations l'analyseur lexical (TOKEN_MGR_DEC)
4. Une section optionnelle pour les caractères à ignorer par l'analyseur syntaxique (SKIP)
5. Une ou plusieurs section (TOKEN) pour déclarer les unités lexicales
6. Une section pour définir les non terminaux

SECTION 1: OPTIONS

```
options //1- quelques options avec leurs valeurs par défaut:
{
  STATIC = true; //valeur par défaut true
  LOOKAHEAD= 1; CHOICE_AMBIGUITY_CHECK = 2; //aide au choix du nombre de lookahead pour les productions de la forme // A |
  OTHER_AMBIGUITY_CHECK = 1; //aide au choix du nombre de LOOKAHEAD pour les production de la forme (A)*, (A)+ , (A)?
  IGNORE_CASE = false; // (default false)
  UNICODE_INPUT = false; // (default false)
  JAVA_UNICODE_ESCAPE = false; // (default false)
}
```

SECTION 2: CLASSE DU PARSEUR

```
PARSER_BEGIN(Parseur)
public class Parseur {
}

PARSER_END(Parseur)
```

Le code java doit être entre les délimiteurs PARSER_BEGIN et PARSER_END, on doit au moins définir le corps de la classe du parseur.

SECTION 3: DÉCLARATIONS POUR L'ANALYSEUR LEXICAL

```
TOKEN_MGR_DECLS : {


}
```

- Cette section optionnelle peut contenir des déclarations java qui pourront être utilisées par l'analyseur lexical.

SECTION 4 : LES CARACTÈRES À IGNORER PAR L'ANALYSEUR LEXICAL (SECTION OPTIONNELLE)

```
SKIP :
{  " "
| "\r"
| "\t"
| "\n"
}
```

SECTION 5: LES TOKENS

- Un token représenté par la classe Token est constitué par l'unité lexicale (attribut kind de type int, dont les valeurs possibles sont définies dans la classe <Parseur>Constants.java) et le lexème (attribut image de type String)
- Exemples de token:
  - a :Token:<a:"a">, unité lexicale: a lexème: a
  - 4 :Token `< Nombre : ([ "0"-"9" ])+ >`, unité lexicale: Nombre, lexème: 4
- Les tokens doivent être définis dans un ou plusieurs blocs tokens:

```
TOKEN : {
  "a"
|
  "b"
|
  < c : "c" > // c est le nom du token
}
```

ACTIONS DANS L'ANALYSEUR LEXICAL

- Dans la section TOKEN, nous pouvons ajouter des actions (code java entre {}), juste après la définition d'un token, exepmple:

```
TOKEN : {
  <Nombre : (["0"-"9"])+> { System.out.println("Nombre:" +
matchedToken.image);}
 }
```

- matchedToken est l'objet Token qui correspond à la chaîne capturée par l'expression régulière dans le flux en entrée.

- Nous pouvons associer aussi une action à tous les tokens, en activant la méthode CommonTokenAction dans les options et en la définissant dans la section TOKEN_MGR_DECLS:
  1. Dans les options: `COMMON_TOKEN_ACTION = true;`
  2. Dans `TOKEN_MGR_DECLS`

```
TOKEN_MGR_DECLS : {
  CommonTokenAction(Token t) {
    System.out.println("Token reconnu:" + t.image);
  }
}
```

- Le token reconnu est passé en argument à la fonction:

EXPRESSIONS RATIONNELLES (OU RÉGULIERES) ÉTENDUES

- “a” | “b” (les espaces qui ne sont pas entre guillemets sont ignorés)
- (“a”)+ (et non “a”+)
- (“a”)*
- (“a”)?
- ~[“b”,"d"]: un caractère n’appartenant pas à la classe de caractères
- ~[]: représente tous les caractères

- Une expression régulière peut avoir un nom:
  - <UNSIGNED: (["0"-"9"])+>

SECTION 6 LES PRODUCTIONS

- Chaque non terminal est associé à une méthode chargée de reconnaître la partie droite
- En javacc la production B→ b B | c, s'écrit:

```
void B() :
{}
{
  "b" B()
|
  < c > // on peut utiliser soit le nom <c> du token soit la valeur du
token "c"
}
```

ACTIONS SÉMANTIQUES ET GRAMMAIRES ATTRIBUÉES

- Une *action sémantique* est un code associé à une règle de la grammaire et exécutée lorsqu'il y a une prédiction de la règle durant l'analyse.
- Un *attribut* est une donnée associée à un symbole d'une règle de grammaire.
- Par exemple, si le symbole est un terminal, ça pourrait être la valeur du *token (valeur double d'un token réel)*.
- Si le symbole est un non terminal, l'attribut pourrait être une donnée calculée en utilisant les actions sémantiques.
- Une grammaire avec des attributs est appelée une grammaire attribuée.
- Un attribut peut être synthétique (calculé) ou hérité (valeur copiée d'un enfant ou parent).