# Documentación de Conjuntos FIRST y FOLLOW

## Conjuntos FIRST

Los conjuntos FIRST para cada no terminal en la gramática representan los posibles símbolos terminales que pueden aparecer al inicio de una cadena derivada del no terminal.

| No Terminal | Conjunto FIRST |
|---|---|
| PROGRAM | { "{", "class" } |
| METHODS | { "int", "float", "void", "char", "string", "boolean" } |
| PARAMS | { "int", "float", "void", "char", "string", "boolean", ε } |
| BODY | { "int", "float", "void", "char", "string", "boolean", "IDENTIFIER", "return", "while", "if", "do", "for", "switch", "(", "!", "-", "LITERAL", "break", ε } |
| VARIABLE | { "int", "float", "void", "char", "string", "boolean" } |
| ASSIGNMENT | { "IDENTIFIER" } |
| CALL_METHOD | { "IDENTIFIER" } |
| PARAM_VALUES | { "IDENTIFIER", "(", "!", "-", "LITERAL", ε } |
| RETURN | { "return" } |
| WHILE | { "while" } |
| IF | { "if" } |
| DO_WHILE | { "do" } |
| FOR | { "for" } |
| SWITCH | { "switch" } |
| STATEMENT_BLOCK | { "{", "int", "float", "void", "char", "string", "boolean", "IDENTIFIER", "return", "while", "if", "do", "for", "switch", "(", "!", "-", "LITERAL" } |
| EXPRESSION | { "IDENTIFIER", "(", "!", "-", "LITERAL" } |
| X | { "IDENTIFIER", "(", "!", "-", "LITERAL" } |
| Y | { "IDENTIFIER", "(", "!", "-", "LITERAL" } |
| R | { "IDENTIFIER", "(", "-", "LITERAL" } |
| E | { "IDENTIFIER", "(", "-", "LITERAL" } |
| A | { "IDENTIFIER", "(", "-", "LITERAL" } |
| B | { "-", "IDENTIFIER", "(", "LITERAL" } |
| C | { "IDENTIFIER", "(", "LITERAL" } |
| TYPE | { "int", "float", "void", "char", "string", "boolean" } |

## Conjuntos FOLLOW

Los conjuntos FOLLOW para cada no terminal representan los posibles símbolos terminales que pueden aparecer inmediatamente después de una cadena derivada del no terminal.

| No Terminal | Conjunto FOLLOW |
|---|---|
| PROGRAM | { "$" } |
| METHODS | { "int", "float", "void", "char", "string", "boolean", "}" } |
| PARAMS | { ")" } |
| BODY | { "}", "break", "case", "default" } |
| VARIABLE | { ";" } |
| ASSIGNMENT | { ";" } |
| CALL_METHOD | { ";", "+", "-", "*", "/", ")", "<", ">", "==", "!=", "&&", " |
| PARAM_VALUES | { ")" } |
| RETURN | { "}", "break", "case", "default" } |
| WHILE | { "}", ";", "else", "break", "case", "default" } |
| IF | { "}", ";", "else", "break", "case", "default" } |
| DO_WHILE | { "}", ";", "else", "break", "case", "default" } |
| FOR | { "}", ";", "else", "break", "case", "default" } |
| SWITCH | { "}", ";", "else", "break", "case", "default" } |
| STATEMENT_BLOCK | { "}", ";", "else", "while", "break", "case", "default" } |
| EXPRESSION | { ";", ")", ",", ":" } |
| X | { ";", ")", ",", ":" } |
| Y | { " |
| R | { "&&", " |
| E | { "<", ">", "==", "!=", "&&", " |
| A | { "+", "-", "<", ">", "==", "!=", "&&", " |
| B | { "*", "/", "+", "-", "<", ">", "==", "!=", "&&", " |
| C | { "*", "/", "+", "-", "<", ">", "==", "!=", "&&", " |
| TYPE | { "IDENTIFIER" } |

## Estrategias de recuperación de errores

La recuperación de errores en el parser utiliza los conjuntos FIRST y FOLLOW para determinar cómo proceder cuando se encuentra un error:

1. **Modo de pánico**: El parser avanza tokens hasta encontrar uno que esté en el conjunto FIRST o FOLLOW de la regla actual.
2. **Sincronización**:
   - Si se encuentra un token en FIRST, se continúa con el análisis de la regla.
   - Si se encuentra un token en FOLLOW sin haber procesado la regla, se asume que la regla está vacía (ε) y se continúa.
3. **Mensajes de error**: El parser proporciona información sobre el error y la recuperación.

## Ejemplos de recuperación

### Ejemplo 1: Falta un paréntesis de cierre en una condición

java

```java
if (x > 5 { // Error: falta el paréntesis de cierre ')'
    doSomething();
}
```

El parser detecta que falta ')' y se recupera continuando con el análisis del bloque de código.

### Ejemplo 2: Falta un punto y coma

java

```java
int x = 5 // Error: falta ';'
int y = 10;
```

El parser detecta la falta de ';' y se recupera para continuar analizando la siguiente declaración.

### Ejemplo 3: Tipo de dato inválido

java

```java
invalid x = 5; // Error: 'invalid' no es un tipo válido
```

El parser detecta que 'invalid' no pertenece al conjunto FIRST de TYPE y busca el siguiente token en FOLLOW para continuar el análisis.