



Práctica 3



1. Análisis léxico



La clase `CommandParser`

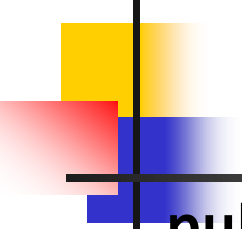
Recordatorio de los *parsers* de la práctica 2

```
public class CommandParser {  
    private final static Command[] commands=  
        {new Help(), new Quit(), new Reset(), new Run(),  
          new AddByteCodeProgram(), new Replace(0)};  
  
    public static Command parse(String line) {  
        //Quitar blancos y dividir en tokens  
        if (tokens.length==0 || tokens.length>2) ...  
        Command cm;  
        for (Command c:commands) {  
            cm=c.parse(tokens);  
            if (cm!=null) return cm;  
        } ...}  
}
```



La classe ByteCodeParser

```
public class ByteCodeParser {  
    private final static ByteCode[] bytecodes=  
        {new Add(), new Sub(),  
         new Mul(), new Div(),  
         new Push(0), new Store(0), new Load(0),  
         new Halt(), new Out(), new Goto(0),  
         new IfEq(0), new IfNEq(0),  
         new IfLe(0), new IfLeq(0)};
```



```
public static ByteCode parse(String line) {  
    //Quitar blancos y dividir en tokens  
    if (tokens.length==0 || tokens.length>2) ...  
    ByteCode ins;  
    for (ByteCode bc:bytecodes) {  
        ins=bc.parse(tokens);  
        if (ins!=null) return ins;  
    }  
    ...  
}
```



La clase `LexicalParser`

- Es el parseador que realiza el análisis léxico
- Parsea el programa fuente, invocando el `InstructionParser` para cada componente del programa fuente, creando un programa parseado que consiste en un array cuyas componentes son las instrucciones ya parseadas

```
■ x = 52  
while x < 0  
    x = 2 * x  
    n = n + 23  
endwhile
```

se parsea en un programa con dos componentes:

```
0: new SimpleAssignment("x", new Number(52))  
1: new While(new Less(new Variable("x"), new Number(0)), new ParsedProgram())  
donde este programa parseado consiste en un array con dos componentes:  
0: new CompoundAssignment("x", "*", new Number(2), new Variable("x"))  
1: new CompoundAssignment("n", "+", new Variable("n"), new Number(23))
```



La clase LexicalParser

- Los atributos de esta clase son:

```
private SourceProgram sProgram;  
private int programCounter;
```

- Su método principal es:

```
public void lexicalParser(  
    ParseredProgram pProgram,  
    String stopKey)  
    throws LexicalAnalysisException{...}
```

que modifica el parámetro **pProgram** a fin de que contenga el resultado de parsear su **sProgram**, desde la componente **programCounter** hasta que aparezca la palabra **stopKey**. Inicialmente, **Engine** hace la llamada **this.lexParser.lexicalParser(pProgram, "end")**



El método `lexicalParser()` de la clase `LexicalParser`

```
while (programCounter < sProgram.getNumeroInstrucciones()
    && ...) {
    ...
    String line = sProgram.getInstruction(programCounter);
    ...
    if (line.equalsIgnoreCase(stopKey)) stop = true;
    else {
        Instruction instruction =
            ParserInstruction.parse(line, this);
        ...//Si instruction no es null
        pProgram.addInstruction(instruction);
        //Aumentar programCounter
    }
}
```




Otros parseadores

- La clase **InstructionParser** parsea las instrucciones
- La interfaz **Instruction** es implementada por las clases **SimpleAssignment**, **While**, **If**, **Write**, **CompoundAssignment** y **Return**
- La clase abstracta **ConditionParser** parsea las condiciones booleanas que aparecen en las instrucciones **while** e **if**
- De la clase abstracta **Condition** heredan las clases **Less**, **LessEq**, **Equal** y **NotEqual**



La interfaz **Instruction** y la clase **InstructionParser**

```
■ interface Instruction {  
    Instruction lexParse(String[] words,  
        LexicalParser lexer);  
    void compile(Compiler compiler) throws ArrayException;  
}  
  
■ class InstructionParser {  
    private final static Instruction[] instructions={  
        new SimpleAssignment(), new CompoundAssignment(),  
        new Write(), new Return(), new While(), new IfThen()  
    }  
    ...  
}
```



La clase abstracta **Condition**

```
abstract class Condition {  
    private Term term1, term2;  
    protected ConditionalJump cj; //para la compilación  
    ...  
    public Condition parse(String t1, String op, String t2,  
        LexicalParser parser) {  
        this.term1=TermParser.parse(t1);  
        this.term2=TermParser.parse(t2);  
        ...  
        return parseOp(term1, op, term2, parser);  
    }  
  
    protected abstract Condition parseOp(Term t1, String op, Term t2,  
        LexicalParser lexParser);  
    ...  
}
```



La classe `ConditionParser`

```
class ConditionParser {  
    private final static Condition[]  
        conditions={new Less(), new LessEq(),  
        new Equal(), new NotEqual()};  
    ...  
}
```



La clase `TermParser`

La clase `TermParser` parsea los términos que aparecen en las asignaciones, simples y compuestas, y en las condiciones del `if` y del `while`

```
■ class TermParser {  
    private final static Term[] terms =  
        {new Variable(""), new Number(0)};  
    public static Term parse(String st) {  
        Term tm;  
        for (Term t:terms) {  
            tm=t.parse(st);  
            if (tm!=null) return tm;  
        }  
        return null;  
    }  
}
```



La classe **Variable**

```
class Variable implements Term {  
    private String varName;  
    ...  
    @Override  
    public Term parse(String term) {  
        if (term.length()!=1) return null;  
        else {  
            char name = term.charAt(0);  
            if ('a'<=name && name<='z')  
                return new Variable(term);  
            else return null;  
        }  
    }  
}
```



La clase SimpleAssignment

- Tiene dos atributos:

```
private String varName;  
private Term rhs;
```

- Su método parseador es:

```
public Instruction lexParse(String[] words,  
                           LexicalParser lexParser) {  
    if (words.length!=3) return null;  
    ...  
    if (words[1]!="=") return null;  
    Term rhs=TermParser.parse(words[2]);  
    if (rhs==null) return null;  
    //lexParser aumenta su contador  
    return new SimpleAssignment(varName, rhs);  
    ...  
}
```



La classe **CompoundAssignment**

- **class CompoundAssignment implements
Instruction {**

```
private String varName;  
private String operator;  
private Term term1, term2;
```

```
...
```

```
}
```




La clase While

Tiene dos atributos:

```
private Condition condition;
```

```
private ParsedProgram body;
```

- Su método parseador es:

```
public Instruction lexParse(String[] words,  
                           LexicalParser lexer) {
```

```
...
```

```
//El parseo de la condición devuelve cond
```

```
ParsedProgram wb = new ParsedProgram();
```

```
lexer.lexicalParser(wb, "ENDWHILE");
```

```
lexer.increaseProgramCounter();
```

```
return new While(cond, wb);
```

```
}
```



La clase **If**

- Es análoga a la clase **While**