

```
package analyzer;

// TODO add the default to the switches
import java.io.IOException;

import ast.Elem;

/**
 * Class that develops a Regular Expression's syntax parser
 * @author Sergio Alcocer
 */
public class Parser {

    private Token nextToken;
    private Lexer scanner;

    /**
     * Constructor of the class
     * @param filename file name (with full path) to be analyzed
     * @throws IOException if something goes wrong
     */
    public Parser(String filename) throws IOException {
        System.out.println("Analysing file: " + filename);
        System.out.println("=====");
        this.scanner = new Lexer(filename);
    }

    /**
     * @param kind Type of the Token to be burned
     * @throws ParseException when the kind to be burned and the actual Token doesn't match
     */
    private void burn(int kind) {
        if(nextToken.getKind() == kind){
            nextToken = scanner.getNextToken();
        }else{
            System.out.println("Error: " + nextToken.getLexeme());
        }
    }

    /**
```

```
* Analyzes the file to create the Abstract Syntax Tree
* @return ast.File object with the structure of the file if everything went right<br>otherwise, <b>null</b>
* @throws Exception
*/
public ast.File Analyze() throws Exception{
    ast.File returned;
    nextToken = scanner.getNextToken();
    returned = parseFile();

    if(nextToken.getKind() == Token.EOF){
        System.out.println("ok");
        return returned;
    }else{
        System.out.println("Error");
        return null;
    }
}

private ast.File parseFile() throws Exception{
    ast.File returned = new ast.File();
    returned = parsePackage(returned);
    returned = parseImports(returned);
    returned = parseClass(returned);
    return returned;
}

private ast.File parsePackage(ast.File inherit) throws Exception{
    ast.File returned = inherit;
    switch(nextToken.getKind()){
        case Token.PACKAGE:
            burn(Token.PACKAGE);
            returned.setPackage(nextToken.getLexeme());
            burn(Token.VALUE);
            burn(Token.SC);
            break;
        case Token.IMPORT:
        case Token.PUBLIC:
        case Token.CLASS:
            break;
    }
}
```

```
    }  
    return returned;  
}  
  
private ast.File parseImports(ast.File inherit) throws Exception{  
    ast.File returned = inherit;  
    switch(nextToken.getKind()){  
        case Token.IMPORT:  
            burn(Token.IMPORT);  
            returned.addImport(nextToken.getLexeme());  
            burn(Token.VALUE);  
            burn(Token.SC);  
            returned = parseImports(returned);  
            break;  
        case Token.PUBLIC:  
        case Token.CLASS:  
            break;  
    }  
    return returned;  
}
```

```
private ast.File parseClass(ast.File inherit) throws Exception{  
    ast.File returned = inherit;  
    switch(nextToken.getKind()){  
        case Token.PUBLIC:  
            parseClassModifier();  
            burn(Token.CLASS);  
            returned.fileclass.setName(nextToken.getLexeme());  
            burn(Token.VALUE);  
            burn(Token.OPEN_CURLY);  
            returned = parseClassBody(returned);  
            burn(Token.CLOSE_CURLY);  
            break;  
        case Token.CLASS:  
            burn(Token.CLASS);  
            returned.fileclass.setName(nextToken.getLexeme());  
            burn(Token.VALUE);  
            burn(Token.OPEN_CURLY);  
            returned = parseClassBody(returned);
```

```
        burn(Token.CLOSE_CURLY);
        break;
    }
    return returned;
}

private void parseClassModifier() throws Exception{
    switch(nextToken.getKind()){
        case Token.PUBLIC:
            burn(Token.PUBLIC);
            break;
        case Token.CLASS:
            break;
    }
}

private ast.File parseClassBody(ast.File inherit) throws Exception{
    ast.File returned = inherit;
    switch(nextToken.getKind()){
        case Token.PUBLIC:
        case Token.PRIVATE:
            int access = parseAccess();
            String type = nextToken.getLexeme();
            burn(Token.VALUE);
            ast.Elem body2 = parseClassBody2();
            if(body2.getType() == ast.Elem.T_ATTRIBUTE){
                ((ast.Attribute)body2).setAccess(access);
                ((ast.Attribute)body2).setAttributeType(type);
                returned.fileclass.addAttribute((ast.Attribute)body2);
            }else if (body2.getType() == ast.Elem.T_METHOD){
                ((ast.Method)body2).setAccess(access);
                ((ast.Method)body2).setReturnedType(type);
                returned.fileclass.addMethod((ast.Method)body2);
            }
            returned = parseClassBody(returned);
            break;
        case Token.CLOSE_CURLY:
            break;
    }
    return returned;
}
```

```
}

private ast.Elem parseClassBody2() throws Exception{
    ast.Elem returned = null;
    switch(nextToken.getKind()){
    case Token.SC:
    case Token.OPEN_BRACK:
        returned = parseClassBody3();
        if(returned.getType() == ast.Elem.T_METHOD){
            ((ast.Method)returned).setAsConstructor();
        }
        break;
    case Token.VALUE:
        String elemName = nextToken.getLexeme();
        burn(Token.VALUE);
        returned = parseClassBody3();
        //if(returned.getType() == ast.Elem.T_METHOD){
        //    ((ast.Method)returned).setName(elemName);
        //}else if (returned.getType() == ast.Elem.T_ATTRIBUTE){
        //    ((ast.Attribute)returned).setName(elemName);
        //}
        returned.setName(elemName);
        break;
    }
    return returned;
}
```

```
private ast.Elem parseClassBody3() throws Exception{
    ast.Elem returned = null;
    switch(nextToken.getKind()){
    case Token.SC:
        returned = new ast.Attribute();
        burn(Token.SC);
        break;
    case Token.OPEN_BRACK:
        returned = new ast.Method();
        burn(Token.OPEN_BRACK);
        returned = parseParams((ast.Method)returned);
        burn(Token.CLOSE_BRACK);
    }
```

```
        burn(Token.OPEN_CURLY);
        parseBodyMethod();
        burn(Token.CLOSE_CURLY);
        break;
    }
    return returned;
}

private ast.Method parseParams(ast.Method inherit) throws Exception{
    ast.Method returned = inherit;
    switch(nextToken.getKind()){
    case Token.VALUE:
        returned.addParam(nextToken.getLexeme());
        burn(Token.VALUE);
        burn(Token.VALUE);
        returned = parseParams(returned);
        break;
    case Token.CLOSE_BRACK:
        break;
    }
    return returned;
}

private int parseAccess() throws Exception{
    int returned = -1;
    switch(nextToken.getKind()){
    case Token.PUBLIC:
        burn(Token.PUBLIC);
        returned = Elem.PUBLIC;
        break;
    case Token.PRIVATE:
        burn(Token.PRIVATE);
        returned = Elem.PRIVATE;
        break;
    }
    return returned;
}

private void parseBodyMethod() throws Exception{
    int open_curly = 1;
```

```
        while(open_curly > 1 || nextToken.getKind() != Token.CLOSE_CURLY){
            switch(nextToken.getKind()){
                case Token.OPEN_CURLY:
                    open_curly++;
                    break;
                case Token.CLOSE_CURLY:
                    open_curly--;
                    break;
            }
            burn(nextToken.getKind());
        }
    }

    /**
     * Small main to perform tests
     * @param param nothing useful
     */
    public static void main(String[] param){
        Parser p;
        try {
            p = new Parser("book.java");
            ast.File returned = p.Analyze();
            returned.getClass();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```