

Contents

1	Basic Test Results	2
2	README	3
3	oop/ex7/main/AssignmentLine.java	6
4	oop/ex7/main/CompilationException.java	8
5	oop/ex7/main/IllegalReturnLocationUsageException.java	9
6	oop/ex7/main/IllegalSyntaxException.java	10
7	oop/ex7/main/Parser.java	11
8	oop/ex7/main/ReturnLine.java	14
9	oop/ex7/main/SJavaLine.java	16
10	oop/ex7/main/SJavaSyntax.java	17
11	oop/ex7/main/Sjavac.java	20
12	oop/ex7/main/parameters/IllegalNameException.java	22
13	oop/ex7/main/parameters/IllegalParameterException.java	23
14	oop/ex7/main/parameters/MethodCalling.java	24
15	oop/ex7/main/parameters/MethodCallingException.java	26
16	oop/ex7/main/parameters/MethodCallingNotExistException.java	27
17	oop/ex7/main/parameters/Operandable.java	28
18	oop/ex7/main/parameters/Parameter.java	29
19	oop/ex7/main/parameters/ParameterCompilationException.java	34
20	oop/ex7/main/parameters/ParameterNotExistException.java	35
21	oop/ex7/main/scopes/ConditionalScope.java	36
22	oop/ex7/main/scopes/MethodScope.java	38

23	oop/ex7/main/scopes/MismatchConditionTypeException.java	40
24	oop/ex7/main/scopes/Scope.java	41
25	oop/ex7/main/parameters/variables/BooleanType.java	42
26	oop/ex7/main/parameters/variables/CharType.java	44
27	oop/ex7/main/parameters/variables/DoubleType.java	46
28	oop/ex7/main/parameters/variables/GenericType.java	48
29	oop/ex7/main/parameters/variables/IllegalArrayLocationException.java	50
30	oop/ex7/main/parameters/variables/IllegalVariableSyntaxException.java	51
31	oop/ex7/main/parameters/variables/IllegalVariableTypeException.java	52
32	oop/ex7/main/parameters/variables/IntegerType.java	53
33	oop/ex7/main/parameters/variables/MethodCallingTypeMismatchException.java	55
34	oop/ex7/main/parameters/variables/NotOperandableMethodCallingException.java	56
35	oop/ex7/main/parameters/variables/NotOperandableVariableException.java	57
36	oop/ex7/main/parameters/variables/StringType.java	58
37	oop/ex7/main/parameters/variables/UninitializedVariableException.java	60
38	oop/ex7/main/parameters/variables/Variable.java	61
39	oop/ex7/main/parameters/variables/VariableAlreadyExistException.java	66
40	oop/ex7/main/parameters/variables/VariableCompilationException.java	67
41	oop/ex7/main/parameters/variables/VariableFactory.java	68
42	oop/ex7/main/parameters/variables/VariableNotExistException.java	70
43	oop/ex7/main/parameters/variables/VariableTypeMismatchException.java	71
44	oop/ex7/main/parameters/variables/VoidMethodException.java	72

1 Basic Test Results

```
1 Logins: roeia1
2
3
4
5 compiling with
6     javac -cp ./cs/course/2013/oop/lib/junit4.jar *.java oop/ex7/main/*.java
7
8
9 tests output :
10     Perfect!
```

2 README

```
1 roeial
2 yinnonbar
3
4
5 File Description
6 -----
7
8     The file consists 4 packages :
9
10    1. main - which contains :
11
12        AssignmentLine - This class represents an s-java assignment line. (X = Y)
13
14        CompilationException - An exception for compilation errors.
15
16        IllegalReturnLocationUsageException - This exception extends the
17        CompilationException and being thrown when an illegal value is given as
18        the return location.
19
20        IllegalSyntaxException - This exception extends the Exception class and
21        being thrown when an illegal syntax line is given.
22
23        Parser - This class is the parser. It is reading the lines in the file
24        and turning it to objects while checking each line syntax.
25
26        ReturnLine - This class represents a return line
27        (Consist the "return" expression). Extending the SjavaLine class.
28
29        Sjavac - This class is the main class of program. This is the compiler, it
30        is taking the lines from the parser and prints the matching message.
31
32        SJavaLine - This class represents an s-java line in the file being read.
33
34        SJavaSyntax - This class represents all the valid syntax for a line in
35        s-java file.
36
37    2. main.parameters - which contains :
38
39        IllegalNameException - This exception indicates that a name is a keyword.
40
41        IllegalParameterException - This exception indicates that there is a
42        parameter syntax exception.
43
44        MethodCalling - This class represents a s-java line method calling.
45        Extending the Parameter class.
46
47        MethodCallingException - This exception indicates that there is a
48        compilation error in a method calling.
49
50        MethodCallingNotExistException - This exception indicates that the method
51        calling isn't exist, a compilation error.
52
53        Operandable - This interface indicates that a parameter is operandable.
54
55        Parameter - This abstract class is the parameter class, extends SJavaLine
56        and designed to work with initializing variables and method calling.
57
58        ParameterCompilationException - This exception indicates that there is a
59        compilation error in a parameter.
```

ParameterNotExistException - This exception indicates that a parameter isn't exist, a compilation error.

3. main.parameters.variables - which contains :

BooleanType - This class is the BooleanType extends the variable class (true\false).

CharType - This class is the CharType extends the variable class.

DoubleType - This class is the DoubleType extends the variable class (5,5.2), and it is also implements Operandable interface (5+3.2).

GenericType - This class is the GenericType variable, extends the variable class. GenericType is a variable which his type is unknown in the syntax check phase.

IllegalArrayLocationException - This exception indicates there is an error in the array location (abc[-1]).

IllegalVariableSyntaxException - This exception indicates that there is a syntax error in a variable.

IllegalVariableTypeException - This exception indicates that there is a syntax error in the variable type.

IntegerType - This class represents the integer type variable, extends from double type.

MethodCallingTypeMismatchException - This exception indicates that there is a method calling return variable mismatch to the assigned variable.

NotOperandableMethodCallingException - This exception indicates that a method calling return variable isn't operandable while using it with operand.

NotOperandableVariableException - This exception indicates that a variable isn't operandable while using it with operand.

StringType - This class represents a string variable, extends from variable class.

UninitializedVariableException - This exception indicates that an uninitialized variable used for assignment, compilation error.

Variable - This abstract class represent all of the legitimate variables in s-java. It extends the parameter class.

VariableAlreadyExistException - This exception extends the CompilationException and being thrown when a variable is already exist.

VariableCompilationException - This exception indicates that there is a compilation error with the variable.

VariableFactory - This class represents the variable factory.

VariableNotExistException - This exception indicates that a variable isn't exist, compilation error.

VariableTypeMismatchException - This exception indicates that there is a variable mismatch to the assigned variable.

VoidMethodException - This exception indicates that there is a method calling returning void used in assignment.

4. main.scopes - which contains :

```

128     ConditionalScope - This class represents a scope containing a condition.
129     (if / while scope). Extending the Scope class.
130
131     MethodScope - This class represents a method scope in the s-java file.
132     Extends the Scope class.
133
134     MismatchConditionTypeException - This exception indicates that the
135     condition is not a boolean type.
136
137     Scope - This class represents a scope in the s-java file.
138
139     README - This file.
140
141 Design
142 -----
143
144     In this project, since we have been asked to build a compiler,
145     there is a total separation between the
146     syntax checking and the compilation checking (the "IsValid" method).
147     Basically, we created a tree when each node is a scope.
148     There are 2 different scopes that inheriting from the scope class,
149     the method scope and the simple scope (an if/while scope).
150     Each scope also have an array list of SJavaLine object,
151     this line represent everything but a scope.
152     First, in the syntax process we created a switch for each line regex
153     we could encounter when reading the lines from the file.
154     This switch knows what regex the line match with an enum that check it.
155     (this enum is in the SJavaSyntax class that also contains most of the
156     compiler regexes)
157     Once we know what regex this line match to, we created an SJavaLine object
158     and added it to the relevant scope SJavaLine array.
159     After all the lines in the file converted to objects and the syntax
160     process is complete, we checked each SJavaLine with an abstract method
161     in the SJavaLine abstract class "IsValid".(each line option class inherit
162     from the abstract class SJavaLine)
163     We separated the packages in this project inside the main package
164     "filesript" to a scope package and an SJavaLine package.
165     We chose to used exceptions in the error handling issue because of the fact
166     that in this way we can use hierarchy in the exception catching.
167     Since we are using different exception classes with inheritance,
168     when an exception being catch we can throw a precise error message
169     indicating the mistake. (we also implemented the print stack trace in each
170     exception with the constructor of the message)
171     In case we need to add a new type of variable we just need to implement it
172     while extending from the Variable abstract class.(Obviously we will need
173     to add the new type name to the SJavaSyntax different regexes).
174     Implementing an if-else block would be quite easy.
175     Since we made a simple block class containing a condition representing
176     both if and while blocks, we will need to separate them to different
177     classes.
178     When we encounter an else scope we will create an object of a Scope type
179     and the syntax check would be checking in the father array of child scopes
180     if before the this else scope there is an instance of an if scope.
181     We used variable declaration regex and declaration method regex with
182     groups to get all the data required for the constructor of each class.
183     In variable declaration regex the groups are the following: type, name,
184     equal sign(doesn't have to appear), value(doesn't have to appear).
185     In declaration method regex the groups are the following: return type,
186     name, all the parameters(doesn't have to appear).
187     Additionally, we did an inheritance from integer to double because that
188     in the assignment integer can be assigned to double but not the other way
189     around. We also did an Operandable interface indicating if a parameter
190     can be with operand or not.
191     Because of this separation from parsing to compilation in this project,
192     We really think the design and implementation stands out from the rest.
193     (Did I hear a bonus? :P)

```

3 oop/ex7/main/AssignmentLine.java

```
1 package oop.ex7.main;
2
3 import java.util.ArrayList;
4
5 import oop.ex7.main.parameters.Parameter;
6 import oop.ex7.main.parameters.ParameterCompilationException;
7 import oop.ex7.main.parameters.variables.GenericType;
8 import oop.ex7.main.parameters.variables.IntegerType;
9 import oop.ex7.main.parameters.variables.Variable;
10 import oop.ex7.main.scopes.Scope;
11
12 /**
13  * This class represents an s-java assignment line. ( $X = Y$ )
14  *
15  * @author roeial
16  *
17  */
18 public class AssignmentLine extends SJavaLine {
19
20     private GenericType assignedGeneric;
21     private Parameter assigningParameter;
22
23     /**
24      * A data constructor.
25      *
26      * @param assignedVariable
27      *      - the name of the variable being assigned.
28      * @param assignedValue
29      *      - the assignment value.
30      * @throws IllegalArgumentException
31      *      if there is a syntax error in the assignment line.
32      */
33     public AssignmentLine(String assignedVariable, String assignedValue)
34         throws IllegalArgumentException {
35         Parameter assignedParameter =
36             Parameter.createOneParameter(assignedVariable);
37         if (assignedParameter instanceof GenericType) {
38             this.assignedGeneric = (GenericType) assignedParameter;
39         } else {
40             throw new IllegalArgumentException(
41                 "Illegal assignment line syntax error");
42         }
43         assigningParameter = Parameter.createParameter(assignedValue);
44     }
45
46     public void isValid(Scope currentScope)
47         throws ParameterCompilationException {
48         // Checking the array location if it is array
49         if (assignedGeneric.isArray()) {
50             ArrayList<Parameter> arrayLocationParameters =
51                 new ArrayList<Parameter>();
52             // Creating an array from the parameters of the array location
53             arrayLocationParameters.add(assignedGeneric
54                 .getArrayLocationParameter());
55             if (!assignedGeneric.getArrayLocationParameter()
56                 .getParameterList().isEmpty()) {
57                 arrayLocationParameters.add(assignedGeneric
58                     .getArrayLocationParameter().getParameterList().get(0));
59             }
60         }
61     }
62 }
```

```

60         for (Parameter currArrayLocationParameter :
61             arrayLocationParameters) {
62             // If the parameter is a generic or raw and not being integer,
63             // or a method calling not returning integer throw exception
64             new IntegerType().checkAssignment(currArrayLocationParameter,
65                 currentScope);
66         }
67     }
68     // Finding the assigned generic variable
69     Variable assignedVariable =
70         this.assignedGeneric.find(currentScope, false);
71     // Checking if its a valid assignment
72     assignedVariable
73         .checkAssignment(this.assignedParameter, currentScope);
74
75 }
76
77 public Variable getAssignedGeneric() {
78     return assignedGeneric;
79 }
80
81 public Parameter getAssigningParameter() {
82     return assigningParameter;
83 }
84 }

```


4 oop/ex7/main/CompilationException.java

```
1  package oop.ex7.main;
2
3  /**
4   * An exception for compilation errors.
5   *
6   * @author roeial
7   *
8   */
9  public class CompilationException extends Exception {
10
11     /**
12      * super the constructor with error message.
13      */
14     public CompilationException(String errorMessage) {
15         super(errorMessage);
16     }
17
18     private static final long serialVersionUID = 1L;
19
20 }
```

5 oop/ex7/main/IllegalReturnLocationUsageException.

```
1 package oop.ex7.main;
2
3 /**
4  * This exception extends the CompilationException and being thrown when an
5  * illegal value is given as the return location.
6  *
7  * @author roeial
8  *
9  */
10 public class IllegalReturnLocationUsageException extends CompilationException {
11
12     /**
13      * super the constructor with error message.
14      */
15     public IllegalReturnLocationUsageException(String errorMessage) {
16         super(errorMessage);
17     }
18
19     private static final long serialVersionUID = 1L;
20
21 }
```

6 oop/ex7/main/IllegalSyntaxException.java

```
1 package oop.ex7.main;
2
3 /**
4  * This exception extends the Exception class and being thrown when an illegal
5  * syntax line is given.
6  *
7  * @author roeial
8  *
9  */
10 public class IllegalSyntaxException extends Exception {
11
12     /**
13      * super the constructor with error message.
14      */
15     public IllegalSyntaxException(String errorMessage) {
16         super(errorMessage);
17     }
18
19     private static final long serialVersionUID = 1L;
20
21 }
```

7 oop/ex7/main/Parser.java

```
1 package oop.ex7.main;
2
3 import java.io.FileReader;
4 import java.io.IOException;
5 import java.io.LineNumberReader;
6
7 import oop.ex7.main.parameters.MethodCalling;
8 import oop.ex7.main.parameters.variables.VariableFactory;
9 import oop.ex7.main.scopes.ConditionalScope;
10 import oop.ex7.main.scopes.MethodScope;
11 import oop.ex7.main.scopes.Scope;
12
13 /**
14  * This class is the parser. It is reading the lines in the file and turning it
15  * to objects while checking each line syntax.
16  *
17  * @author roeial
18  *
19  */
20 public class Parser {
21     /**
22      * Parsing the file with the given file location, checking for valid s-java
23      * syntax.
24      *
25      * @param fileLocation
26      *      - the file location.
27      * @throws IllegalSyntaxException
28      *      if there is a syntax error.
29      */
30     public static void Parse(String fileLocation)
31         throws IllegalSyntaxException, IOException {
32         LineNumberReader lineReader =
33             new LineNumberReader(new FileReader(fileLocation));
34         Scope currentScope = Sjavac.MAIN_SCOPE;
35         String currLine = lineReader.readLine();
36         // reading the given file while the current line is not null, means
37         // there's a line to read, dividing it to cases with switch
38         // according to the matching regex
39         while (currLine != null) {
40             SJavaSyntax.Syntax syntax = SJavaSyntax.getSyntax(currLine);
41             switch (syntax) {
42                 case BLANK_LINE:
43                     break;
44                 case COMMENT:
45                     break;
46                 case END_OF_SCOPE:
47                     if (currentScope == null) {
48                         throw new IllegalSyntaxException("Illegal } usage error");
49                     } else {
50                         currentScope = currentScope.getFatherScope();
51                     }
52                     break;
53                 case DECLARATION_METHOD:
54                     // Method calling
55                     if (syntax.getLineMatcher().group(
56                         SJavaSyntax.METHOD_RETURN_TYPE_GROUP) == null) {
57                         // Checking if called in main scope and throw exception
58                         // if it does since it's an illegal calling for method
59                         // at the main scope.
```

```

60         if (currentScope == Sjavac.MAIN_SCOPE) {
61             throw new IllegalArgumentException(
62                 "Method calling outside a method "
63                 + "without assignment to a member");
64         } else {
65             currentScope.getLineList().add(
66                 new MethodCalling(syntax.getLineMatcher().group(
67                     SJavaSyntax.METHOD_NAME_GROUP), syntax
68                     .getLineMatcher().group(
69                         SJavaSyntax.METHOD_PARAMETERS_GROUP)));
70         }
71         // Method declaration
72     } else {
73         // Checking if declared in main scope
74         if (currentScope == Sjavac.MAIN_SCOPE) {
75             Sjavac.MAIN_SCOPE
76                 .getChildScopeList()
77                 .add(
78                     new MethodScope(
79                         Sjavac.MAIN_SCOPE,
80                         syntax.getLineMatcher().group(
81                             SJavaSyntax.METHOD_RETURN_TYPE_GROUP),
82                         syntax
83                             .getLineMatcher()
84                             .group(
85                                 SJavaSyntax
86                                     .METHOD_RETURN_ARRAY_SIGN_GROUP),
87                         syntax.getLineMatcher().group(
88                             SJavaSyntax.METHOD_NAME_GROUP),
89                         syntax.getLineMatcher().group(
90                             SJavaSyntax.METHOD_PARAMETERS_GROUP)));
91             currentScope =
92                 currentScope.getChildScopeList().get(
93                     currentScope.getChildScopeList().size() - 1);
94         } else {
95             throw new IllegalArgumentException(
96                 "Method declaration not in main scope error");
97         }
98     }
99     break;
100 case RETURN:
101     currentScope.getLineList().add(
102         new ReturnLine(syntax.getLineMatcher().group(
103             SJavaSyntax.RETURN_VALUE_GROUP)));
104     break;
105 case SIMPLE_SCOPE:
106     // Checking if the simple scope opens in the main scope
107     // (if / while usage in main scope)
108     if (currentScope == Sjavac.MAIN_SCOPE) {
109         throw new IllegalArgumentException(
110             "if/while usage out of a method scope error");
111         // else adding that simple scope as a child to the
112         // current
113         // scope.
114     } else {
115         currentScope.getChildScopeList().add(
116             new ConditionalScope(currentScope, syntax
117                 .getLineMatcher().group(
118                     SJavaSyntax.SIMPLE_SCOPE_CONDITION_GROUP)));
119         currentScope =
120             currentScope.getChildScopeList().get(
121                 currentScope.getChildScopeList().size() - 1);
122     }
123     break;
124 case VARIABLE:
125     // in case the line matching to a variable send it to the
126     // variable factory.
127     currentScope.getLineList().add(

```

```

128         VariableFactory.createVariable(syntax.getLineMatcher()
129             .group(SJavaSyntax.VARIABLE_TYPE_GROUP), syntax
130             .getLineMatcher().group(
131                 SJavaSyntax.VARIABLE_ARRAY_SIGN_GROUP), syntax
132             .getLineMatcher().group(
133                 SJavaSyntax.VARIABLE_NAME_GROUP), syntax
134             .getLineMatcher().group(
135                 SJavaSyntax.VARIABLE_VALUE_GROUP)));
136         break;
137     case ASSIGNMENT:
138         // in case of an assignment (=) sending the current line to
139         // assignment line constructor.
140         currentScope.getLineList().add(
141             new AssignmentLine(syntax.getLineMatcher().group(
142                 SJavaSyntax.ASSIGNMENT_VARIABLE_GROUP), syntax
143                 .getLineMatcher().group(
144                     SJavaSyntax.ASSIGNMENT_VALUE_GROUP)));
145         break;
146     case CALLING_METHOD:
147         currentScope.getLineList().add(
148             new MethodCalling(syntax.getLineMatcher().group(
149                 SJavaSyntax.METHOD_CALLING_NAME_GROUP), syntax
150                 .getLineMatcher().group(
151                     SJavaSyntax.METHOD_CALLING_PARAMETERS_GROUP)));
152         break;
153     default:
154         // in case that the line is not matching to any of the
155         // cases above throw an illegal syntax exception.
156         throw new IllegalSyntaxException("Illegal syntax line error");
157     }
158     currLine = lineReader.readLine();
159 }
160 lineReader.close();
161 }
162 }
163 }

```

8 oop/ex7/main/ReturnLine.java

```
1  package oop.ex7.main;
2
3  import java.util.ArrayList;
4
5  import oop.ex7.main.parameters.IllegalParameterException;
6  import oop.ex7.main.parameters.Parameter;
7  import oop.ex7.main.parameters.variables.Variable;
8  import oop.ex7.main.scopes.MethodScope;
9  import oop.ex7.main.scopes.Scope;
10
11  /**
12   * This class represents a return line (Consist the "return" expression).
13   * Extending the SjavaLine class.
14   *
15   * @author roeial
16   *
17   */
18  public class ReturnLine extends SJavaLine {
19
20      private ArrayList<Parameter> parameterList;
21      private boolean isRawArray;
22
23      /**
24       * A data constructor.
25       *
26       * @param returnValue
27       *         - the return value.
28       * @throws IllegalParameterException
29       *         if one of the parameters string isn't match for any type.
30       * @throws IllegalSyntaxException
31       *         if there is a syntax error in the parameters string.
32       */
33      public ReturnLine(String returnValue) throws IllegalParameterException,
34          IllegalSyntaxException {
35          isRawArray = false;
36          parameterList = new ArrayList<Parameter>();
37          if (!returnValue.equals("")) {
38              // Check if the return value is an array
39              if (returnValue.matches(SJavaSyntax.ARRAY_ASSIGNMENT_REGEX)) {
40                  parameterList = Parameter.arrayAssignment(returnValue);
41                  isRawArray = true;
42              } else {
43                  parameterList.add(Parameter.createParameter(returnValue));
44              }
45          }
46      }
47
48      public void isValid(Scope currentScope) throws CompilationException {
49          if (currentScope instanceof MethodScope) {
50              // Checking if the return line in the method scope is not the last
51              // line
52              if (currentScope.getLineList().indexOf(this) != currentScope
53                  .getLineList().size() - 1) {
54                  throw new IllegalReturnLocationUsageException(
55                      "Illegal return location usage inside a method scope");
56              }
57          } else {
58              currentScope = currentScope.getFatherScope();
59              while (!(currentScope instanceof MethodScope)) {
```

```

60         currentScope = currentScope.getFatherScope();
61     }
62 }
63 Variable methodReturnValue =
64     ((MethodScope) currentScope).getReturnValue();
65 // Return line inside a void return_type method
66 if (methodReturnValue == null) {
67     if (!this.parameterList.isEmpty()) {
68         throw new CompilationException(
69             "Return a value in a void return_type method");
70     }
71 } else {
72     if (methodReturnValue.isArray()) {
73         if (this.parameterList.isEmpty() && !this.isRawArray) {
74             throw new CompilationException(
75                 "Mismatch type between the return value and line, "
76                 + "one is array and one isn't");
77         }
78     } else {
79         if (this.isRawArray) {
80             throw new CompilationException(
81                 "Returning raw array when the return "
82                 + "value of the method isn't array");
83         }
84         if (this.parameterList.isEmpty()) {
85             throw new CompilationException(
86                 "No return parameter in a method "
87                 + "that doesn't return void");
88         }
89     }
90     // Checking if every parameter in the return line matching the
91     // return type of the method
92     for (Parameter currParameter : this.parameterList) {
93         methodReturnValue.checkAssignment(currParameter, currentScope);
94     }
95 }
96 }
97
98 public ArrayList<Parameter> getParameterList() {
99     return parameterList;
100 }
101
102 public boolean isRawArray() {
103     return isRawArray;
104 }
105 }

```


9 oop/ex7/main/SJavaLine.java

```
1  package oop.ex7.main;
2
3  import oop.ex7.main.scopes.Scope;
4
5  /**
6   * This class represents an s-java line in the file being read.
7   *
8   * @author roeial
9   *
10  */
11  public abstract class SJavaLine {
12
13      /**
14       * This method check if the current s-java line is compilation valid.
15       *
16       * @param currentScope
17       *         - the current scope.
18       * @throws CompilationException
19       *         if the s-java line has a compilation error.
20       */
21      public abstract void isValid(Scope currentScope)
22          throws CompilationException;
23  }
```

10 oop/ex7/main/SJavaSyntax.java

```
1 package oop.ex7.main;
2
3 import java.util.regex.Matcher;
4 import java.util.regex.Pattern;
5
6 /**
7  * This class represents all the valid syntax for a line in s-java file.
8  *
9  * @author roeial
10  *
11  */
12 public class SJavaSyntax {
13
14     public final static int RETURN_VALUE_GROUP = 1;
15     public final static int VARIABLE_TYPE_GROUP = 1;
16     public final static int VARIABLE_ARRAY_SIGN_GROUP = 2;
17     public final static int VARIABLE_NAME_GROUP = 3;
18     public final static int VARIABLE_VALUE_GROUP = 4;
19     public final static int ASSIGNMENT_VARIABLE_GROUP = 1;
20     public final static int GENERIC_VARIABLE_NAME_GROUP = 1;
21     public final static int GENERIC_ARRAY_LOCATION_GROUP = 2;
22     public final static int ASSIGNMENT_VALUE_GROUP = 4;
23     public final static int METHOD_RETURN_TYPE_GROUP = 1;
24     public final static int METHOD_RETURN_ARRAY_SIGN_GROUP = 2;
25     public final static int METHOD_NAME_GROUP = 3;
26     public final static int METHOD_PARAMETERS_GROUP = 4;
27     public final static int METHOD_CALLING_NAME_GROUP = 1;
28     public final static int METHOD_CALLING_PARAMETERS_GROUP = 2;
29     public final static int TYPE_LOCATION = 1;
30     public final static int ARRAY_LOCATION = 2;
31     public final static int NAME_LOCATION = 3;
32     public final static int EQUAL_SIGN_LOCATION = 4;
33     public final static int VALUE_LOCATION = 5;
34     public final static int SIMPLE_SCOPE_CONDITION_GROUP = 2;
35     public final static String COMMA = "\\s*\\,\\s*";
36     public final static String ARRAY_SIGN = "\\[\\s*\\]";
37     public final static String ASSIGNMENT_ARRAY_SIGN = "\\[\\s*(\\.*)\\s*\\]";
38     public final static String TYPES_REGEX = "(int|boolean|char|double|"
39         + "String)\\s*(" + ARRAY_SIGN + ")?";
40     public final static String METHOD_RETURN_TYPE_REGEX = "(int|boolean|"
41         + "char|double|String|void)\\s*(" + ARRAY_SIGN + ")?";
42     public final static String SIMPLE_SCOPE_NAME = "(if|while)";
43     public final static String SIMPLE_SCOPE_REGEX = "^\\s*"
44         + SIMPLE_SCOPE_NAME + "\\s*\\((\\s*(\\.*)\\s*\\)\\s*\\{\\s*$";
45     public final static String COMMENT_PATTERN = "\\s*/\\.*\\s*";
46     public final static String METHOD_NAME_REGEX = "([A-Za-z]\\w*)";
47     public final static String VARIABLE_NAME_REGEX = "([A-Za-z]\\w*|_\\w+)";
48     public final static String VARIABLE_REGEX = "^\\s*(?:" + TYPES_REGEX
49         + ")\\s*" + VARIABLE_NAME_REGEX + "\\s*(?:\\s*(\\.*)\\s*)?\\s*$";
50     public final static String ASSIGNMENT_REGEX = "^\\s*("
51         + VARIABLE_NAME_REGEX + "(?:\\s*" + ASSIGNMENT_ARRAY_SIGN
52         + ")?)\\s*\\s*=\\s*(\\.*)\\s*\\s*\\s*$";
53     public final static String ARRAY_ASSIGNMENT_REGEX = "^\\{\\s*(\\.*)\\s*\\}\\s*$";
54     public final static String DECLARATION_METHOD_REGEX = "^\\s*"
55         + METHOD_RETURN_TYPE_REGEX + " " + METHOD_NAME_REGEX + "\\s*"
56         + "\\[\\s*(\\.*)\\s*\\]\\s*\\{\\s*\\s*\\s*\\s*\\s*$";
57     public final static String METHOD_CALLING_REGEX = "^\\s*"
58         + METHOD_NAME_REGEX + "\\s*" + "\\[\\s*(\\.*)\\s*\\]\\s*\\s*\\s*$";
59     public final static String OPERAND_REGEX = "\\|\\/\\*|\\-|\\+";
```

```

60 public final static String ANYTHING_BUT_OPERAND_REGEX =
61     "[^\\|\\/|\\*|\\-|\\+|]+";
62 public final static String[] KEYWORDS = new String[] { "void", "int",
63     "double", "boolean", "char", "String" };
64 public final static String BLANK_LINE_PATTERN = "\\s*";
65 public final static String RETURN_REGEX =
66     "\\s*return\\s*(.*)\\s*\\|;\\s*$";
67 public final static String END_OF_SCOPE_REGEX = "\\s*\\}\\s*$";
68
69 /**
70  * This enum containing all the valid syntax for a line in a s-java file.
71  *
72  * @author roeial
73  *
74  */
75 public enum Syntax {
76     VARIABLE(VARIABLE_REGEX),
77     DECLARATION_METHOD(DECLARATION_METHOD_REGEX), CALLING_METHOD(
78     METHOD_CALLING_REGEX), COMMENT(COMMENT_PATTERN), SIMPLE_SCOPE(
79     SIMPLE_SCOPE_REGEX), BLANK_LINE(BLANK_LINE_PATTERN), RETURN(
80     RETURN_REGEX), END_OF_SCOPE(END_OF_SCOPE_REGEX), ASSIGNMENT(
81     ASSIGNMENT_REGEX);
82
83     private final String syntax;
84     private Matcher lineMatcher;
85
86     /**
87      * A data constructor.
88      *
89      * @param syntax
90      *      - the valid syntax of the given line option.
91      */
92     Syntax(String syntax) {
93         this.syntax = syntax;
94     }
95
96     public void setLineMatcher(String line) {
97         this.lineMatcher = Pattern.compile(syntax).matcher(line);
98     }
99
100     public Matcher getLineMatcher() {
101         return lineMatcher;
102     }
103 }
104
105 /**
106  * This static method checking a given s-java line from the file and return
107  * the valid syntax value from the enum.
108  *
109  * @param line
110  *      - the line from the s-java file.
111  * @return The matched value from the enum.
112  * @throws IllegalSyntaxException
113  *      if there is a syntax error in the line.
114  */
115 public static Syntax getSyntax(String line) throws IllegalSyntaxException {
116     for (Syntax syntax : Syntax.values()) {
117         syntax.setLineMatcher(line);
118         if (syntax.getLineMatcher().matches()) {
119             return syntax;
120         }
121     }
122     throw new IllegalSyntaxException("Syntax error");
123 }
124
125 public static void checkIfStartOrEndWithComma(String value)
126     throws IllegalSyntaxException {

```

```
128         if (value.matches(COMMA + ".*") || value.matches(".*?" + COMMA)) {
129             throw new IllegalArgumentException(
130                 "Illegal comma placemnt syntax error");
131         }
132     }
133 }
```

11 oop/ex7/main/Sjavac.java

```
1  package oop.ex7.main;
2
3  import java.io.IOException;
4
5  import oop.ex7.main.scopes.ConditionalScope;
6  import oop.ex7.main.scopes.Scope;
7
8  /**
9   * This class is the main class of program. This is the compiler, it is taking
10   * the lines from the parser and prints the matching message.
11   *
12   * @author roeial
13   *
14   */
15  public class Sjavac {
16
17      public static Scope MAIN_SCOPE;
18
19      /**
20       * This is the main. Prints the matching message - 0 - if the code is legal.
21       * 1 - if the code is illegal. 2- in case of IO errors.
22       *
23       * @param args
24       */
25      public static void main(String[] args) {
26          MAIN_SCOPE = new Scope(null);
27          // the try and catch structure. try if no exception was thrown, means
28          // no error and print 0.
29          try {
30              Parser.Parse(args[0]);
31              CheckScope(MAIN_SCOPE);
32              System.out.println("0");
33              // if met an IllegalSyntaxException or CompilationException than
34              // prints 1 and the matching error message as was given in the
35              // throw exception.
36          } catch (IllegalSyntaxException | CompilationException e) {
37              System.out.println("1");
38              System.err.println(e.getMessage());
39              e.printStackTrace();
40              // another catch for the IO exception.
41          } catch (IOException e) {
42              System.out.println("2");
43          }
44      }
45
46      /**
47       * A static method that in each iteration checks all the s-java line of the
48       * current scope, and later checks all of it's child.
49       *
50       * @param scopeToCheck
51       *        - a given scope.
52       * @throws CompilationException
53       *        if there is a compilation error.
54       */
55      protected static void CheckScope(Scope scopeToCheck)
56          throws CompilationException {
57          // if the scope to check is an if/while scope than check the condition
58          if (scopeToCheck instanceof ConditionalScope) {
59              ((ConditionalScope) scopeToCheck).CheckCondition();
60          }
61      }
62  }
```

```
60     }
61     // sending to is valid for each scope and later for all the child
62     // scopes
63     for (SJavaLine currLine : scopeToCheck.getLineList()) {
64         currLine.isValid(scopeToCheck);
65     }
66     for (Scope currScope : scopeToCheck.getChildScopeList()) {
67         CheckScope(currScope);
68     }
69 }
70 }
```

12 oop/ex7/main/parameters/IllegalNameException.java

```
1  package oop.ex7.main.parameters;
2
3
4  /**
5   * This exception indicates that a name is a keyword.
6   *
7   * @author roeial
8   *
9   */
10 public class IllegalNameException extends IllegalParameterException {
11
12     public IllegalNameException(String errorMessage) {
13         super(errorMessage);
14     }
15
16     /**
17     *
18     */
19     private static final long serialVersionUID = 1L;
20
21 }
```

13 oop/ex7/main/parameters/IllegalParameterException

```
1 package oop.ex7.main.parameters;
2
3 import oop.ex7.main.IllegalSyntaxException;
4
5
6 /**
7  * This exception indicates that there is a parameter syntax exception.
8  *
9  * @author roeial
10  *
11  */
12 public class IllegalParameterException extends IllegalSyntaxException {
13
14     public IllegalParameterException(String errorMessage) {
15         super(errorMessage);
16     }
17
18     /**
19      *
20      */
21     private static final long serialVersionUID = 1L;
22
23 }
```


14 oop/ex7/main/parameters/MethodCalling.java

```
1  package oop.ex7.main.parameters;
2
3  import oop.ex7.main.IllegalSyntaxException;
4  import oop.ex7.main.SJavaSyntax;
5  import oop.ex7.main.Sjavac;
6  import oop.ex7.main.parameters.variables.Variable;
7  import oop.ex7.main.scopes.MethodScope;
8  import oop.ex7.main.scopes.Scope;
9
10 /**
11  * This class represents a s-java line method calling. Extending the Parameter
12  * class.
13  *
14  * @author roeial
15  *
16  */
17 public class MethodCalling extends Parameter {
18
19     private int numOfParameters;
20
21     /**
22      * Constructing a new method calling.
23      *
24      * @param name
25      *         - the name of the method.
26      * @param parameters
27      *         - the string of all the parameters.
28      * @throws IllegalSyntaxException
29      *         if there is a syntax error in the method calling.
30      */
31     public MethodCalling(String name, String parameters)
32         throws IllegalSyntaxException {
33         super(name);
34         numOfParameters = 0;
35         if (!parameters.equals("")) {
36             SJavaSyntax.checkIfStartOrEndWithComma(parameters);
37             String[] parameterStrings = parameters.split(SJavaSyntax.COMMA);
38             for (String currParameter : parameterStrings) {
39                 parameterList.add(Parameter.createParameter(currParameter));
40                 numOfParameters++;
41             }
42         }
43     }
44
45     public void isValid(Scope currentScope)
46         throws ParameterCompilationException, MethodCallingNotExistException {
47         MethodScope calledMethodScope = null;
48         // Searching for the method declaration
49         for (Scope currMethodScope : Sjavac.MAIN_SCOPE.getChildScopeList()) {
50             if (((MethodScope) currMethodScope).getName().equals(
51                 this.getName())) {
52                 calledMethodScope = (MethodScope) currMethodScope;
53                 break;
54             }
55         }
56         if (calledMethodScope == null) {
57             throw new MethodCallingNotExistException(
58                 "Calling a method that don't exist error");
59         }
60     }
61 }
```

```

60     }
61     // Checking the method declaration and calling have the same number of
62     // parameters
63     if (calledMethodScope.getNumOfParameters() != this
64         .getNumOfParameters()) {
65         throw new MethodCallingException(
66             "Mismatch number of variables in method calling error");
67     } else {
68         // Checking if each parameter in the calling match the type
69         // in the declaration
70         for (int currParameterIndex = 0; currParameterIndex <
71             calledMethodScope.getNumOfParameters(); currParameterIndex++) {
72             ((Variable) calledMethodScope.getLineList().get(
73                 currParameterIndex)).checkAssignment(this
74                 .getParameterList().get(currParameterIndex), currentScope);
75         }
76     }
77 }
78
79 public Variable find(Scope currentScope, boolean creationCheck)
80     throws MethodCallingNotExistException {
81     for (Scope currMethodScope : Sjavac.MAIN_SCOPE.getChildScopeList()) {
82         if (((MethodScope) currMethodScope).getName().equals(
83             this.getName())) {
84             return ((MethodScope) currMethodScope).getReturnValue();
85         }
86     }
87     throw new MethodCallingNotExistException("Method isn't found");
88 }
89
90 public int getNumOfParameters() {
91     return numOfParameters;
92 }
93 }

```

15 oop/ex7/main/parameters/MethodCallingException

```
1 package oop.ex7.main.parameters;
2
3 /**
4  * This exception indicates that there is a compilation error in a method
5  * calling.
6  *
7  * @author roeial
8  *
9  */
10 public class MethodCallingException extends ParameterCompilationException {
11
12     public MethodCallingException(String errorMessage) {
13         super(errorMessage);
14     }
15
16     /**
17      *
18      */
19     private static final long serialVersionUID = 1L;
20
21 }
```

16 oop/ex7/main/parameters/MethodCallingNotExistE

```
1 package oop.ex7.main.parameters;
2
3 /**
4  * This exception indicates that the method calling isn't exist, a compilation
5  * error.
6  *
7  * @author roeial
8  *
9  */
10 public class MethodCallingNotExistException extends ParameterNotExistException {
11
12     public MethodCallingNotExistException(String errorMessage) {
13         super(errorMessage);
14     }
15
16     /**
17      *
18      */
19     private static final long serialVersionUID = 1L;
20
21 }
```

17 oop/ex7/main/parameters/Operandable.java

```
1 package oop.ex7.main.parameters;
2
3 /**
4  * This interface indicates that a parameter is operandable.
5  *
6  * @author roeial
7  *
8  */
9 public interface Operandable {
10
11 }
```

18 oop/ex7/main/parameters/Parameter.java

```
1 package oop.ex7.main.parameters;
2
3 import java.util.ArrayList;
4 import java.util.regex.Matcher;
5 import java.util.regex.Pattern;
6
7 import oop.ex7.main.CompilationException;
8 import oop.ex7.main.IllegalSyntaxException;
9 import oop.ex7.main.SJavaLine;
10 import oop.ex7.main.SJavaSyntax;
11 import oop.ex7.main.parameters.variables.BooleanType;
12 import oop.ex7.main.parameters.variables.CharType;
13 import oop.ex7.main.parameters.variables.DoubleType;
14 import oop.ex7.main.parameters.variables.GenericType;
15 import oop.ex7.main.parameters.variables.IntegerType;
16 import oop.ex7.main.parameters.variables.StringType;
17 import oop.ex7.main.parameters.variables.Variable;
18 import oop.ex7.main.scopes.Scope;
19
20 /**
21  * This abstract class is the parameter class, extends SJavaLine and designed to
22  * work with initializing variables and method calling.
23  *
24  * @author roeial
25  *
26  */
27 public abstract class Parameter extends SJavaLine {
28
29     /** The regular expression of two parameters */
30     protected final String TWO_PARAMETERS_ASSIGNMENT_REGEX =
31         "^\\s*\\|-?\\s*(.+)\\s*(?:\\/\\/\\|\\*\\|\\|-\\/\\|+)?\\s*(.*)\\s*$";
32     private final static int FIRST_PARAMETER_GROUP = 1;
33     private final static int SECOND_PARAMETER_GROUP = 2;
34     private final static int ARRAY_ASSIGNMENT_PARAMETERS_GROUP = 1;
35
36     protected ArrayList<Parameter> parameterList;
37     private String name;
38
39     /**
40      * Constructing a raw parameter.
41      */
42     protected Parameter() {
43         parameterList = new ArrayList<Parameter>();
44     }
45
46     /**
47      * Constructing a new parameter.
48      *
49      * @param name
50      *         - the name of the parameter.
51      * @throws IllegalArgumentException
52      *         if the parameter name isn't valid (a keyword).
53      */
54     protected Parameter(String name) throws IllegalArgumentException {
55         this();
56         checkIfNameIsKeyword(name);
57         this.name = name;
58     }
59 }
```

```

60  /**
61   * This enum include all the parameter types.
62   *
63   * @author roeial
64   *
65   */
66  private enum ParameterType {
67      INTEGER(IntegerType.INTEGER_REGEX), DOUBLE(DoubleType.DOUBLE_REGEX),
68      STRING(StringType.STRING_REGEX), CHAR(CharType.CHAR_REGEX), BOOLEAN(
69          BooleanType.BOOLEAN_REGEX), METHOD_CALLING(
70          SJavaSyntax.METHOD_CALLING_REGEX), GENERIC(
71          GenericType.GENERIC_REGEX);
72
73      private final String typeName;
74      private Matcher typeMatcher;
75
76      ParameterType(String type) {
77          this.typeName = type;
78      }
79
80      public void setTypeMatcher(String line) {
81          this.typeMatcher = Pattern.compile(typeName).matcher(line);
82      }
83
84      public Matcher getTypeMatcher() {
85          return typeMatcher;
86      }
87  }
88
89  /**
90   * Getting the type of the parameter string.
91   *
92   * @param parameter
93   *      - the parameter string.
94   * @return The parameter type enum value.
95   * @throws IllegalArgumentException
96   *      if the parameter string isn't match for any type.
97   */
98  public static ParameterType getType(String parameter)
99      throws IllegalArgumentException {
100      for (ParameterType parameterType : ParameterType.values()) {
101          parameterType.setTypeMatcher(parameter);
102          if (parameterType.getTypeMatcher().matches()) {
103              return parameterType;
104          }
105      }
106      throw new IllegalArgumentException("Illegal parameter syntax error");
107  }
108
109  /**
110   * Creating one parameter.
111   *
112   * @param parameter
113   *      - a given string which is the parameter.
114   * @return The created parameter.
115   * @throws IllegalArgumentException
116   *      if the parameter string isn't match for any type.
117   * @throws IllegalArgumentException
118   *      if the parameter is method calling and there is a syntax
119   *      error.
120   */
121  public static Parameter createOneParameter(String parameter)
122      throws IllegalArgumentException, IllegalArgumentException {
123      ParameterType parameterType = getType(parameter);
124      switch (parameterType) {
125          case INTEGER:
126              return new IntegerType();
127          case BOOLEAN:

```

```

128         return new BooleanType();
129     case CHAR:
130         return new CharType();
131     case DOUBLE:
132         return new DoubleType();
133     case STRING:
134         return new StringType();
135     case GENERIC:
136         // Check if the generic is not an array
137         if (parameterType.getTypeMatcher().group(
138             SJavaSyntax.GENERIC_ARRAY_LOCATION_GROUP) == null) {
139             return new GenericType(parameterType.getTypeMatcher().group(
140                 SJavaSyntax.GENERIC_VARIABLE_NAME_GROUP));
141         } else {
142             return new GenericType(parameterType.getTypeMatcher().group(
143                 SJavaSyntax.GENERIC_VARIABLE_NAME_GROUP), parameterType
144                 .getTypeMatcher().group(
145                     SJavaSyntax.GENERIC_ARRAY_LOCATION_GROUP));
146         }
147     case METHOD_CALLING:
148         return new MethodCalling(parameterType.getTypeMatcher().group(
149             SJavaSyntax.METHOD_CALLING_NAME_GROUP), parameterType
150             .getTypeMatcher().group(
151                 SJavaSyntax.METHOD_CALLING_PARAMETERS_GROUP));
152     default:
153         return null;
154     }
155 }
156
157 /**
158  * Creating one parameter, or two parameters (first, operand, second). If
159  * two, adding the second parameter to the first parameter parameterList.
160  *
161  * @param value
162  *      - a given value string.
163  * @return The created parameter.
164  * @throws IllegalParameterException
165  *      if the parameter string isn't match for any type.
166  * @throws IllegalSyntaxException
167  *      if the parameter is method calling and there is a syntax
168  *      error.
169  */
170 public static Parameter createParameter(String value)
171     throws IllegalSyntaxException, IllegalParameterException {
172     Pattern twoParametersPattern =
173         Pattern.compile(TWO_PARAMETERS_ASSIGNMENT_REGEX);
174     Matcher twoParametersMatcher = twoParametersPattern.matcher(value);
175     if (twoParametersMatcher.matches()) {
176         Parameter newParameter =
177             createOneParameter(twoParametersMatcher
178                 .group(FIRST_PARAMETER_GROUP));
179         newParameter.getParameterList().add(
180             createOneParameter(twoParametersMatcher
181                 .group(SECOND_PARAMETER_GROUP)));
182         return newParameter;
183     } else {
184         return createOneParameter(value);
185     }
186 }
187
188 /**
189  * Creating an ArrayList of parameters.
190  *
191  * @param assignmentValue
192  *      - the assignment value string of an array.
193  * @return The ArrayList of parameters.
194  * @throws IllegalSyntaxException
195  *      if there is a syntax error in the parameters string.

```



```

196     * @throws IllegalArgumentException
197     *         if one of the parameters string isn't match for any type.
198     */
199     public static ArrayList<Parameter> arrayAssignment(String assignmentValue)
200     throws IllegalArgumentException, IllegalArgumentException {
201         Pattern arrayAssignmentPattern =
202             Pattern.compile(SJavaSyntax.ARRAY_ASSIGNMENT_REGEX);
203         Matcher arrayAssignmentMatcher =
204             arrayAssignmentPattern.matcher(assignmentValue);
205         // Checking if the array pattern matches
206         if (arrayAssignmentMatcher.matches()) {
207             ArrayList<Parameter> arrayAssignmentParameters =
208                 new ArrayList<Parameter>();
209             // Checking if the array assignment not empty {}
210             if (!arrayAssignmentMatcher.group(
211                 ARRAY_ASSIGNMENT_PARAMETERS_GROUP).equals("")) {
212                 SJavaSyntax.checkIfStartOrEndWithComma(arrayAssignmentMatcher
213                     .group(ARRAY_ASSIGNMENT_PARAMETERS_GROUP));
214                 // Splitting the parameters
215                 String[] parameterStrings =
216                     arrayAssignmentMatcher.group(
217                         ARRAY_ASSIGNMENT_PARAMETERS_GROUP).split(
218                             SJavaSyntax.COMMA);
219                 // Creating each parameter
220                 for (String currArrayParameter : parameterStrings) {
221                     arrayAssignmentParameters
222                         .add(createParameter(currArrayParameter));
223                 }
224             }
225             return arrayAssignmentParameters;
226         } else {
227             // If array pattern don't match
228             throw new IllegalArgumentException("Illegal assignment syntax error");
229         }
230     }
231 }
232
233 /**
234  * Checking if a string is a restricted keyword.
235  *
236  * @param value
237  *         - A given String.
238  * @throws IllegalArgumentException
239  *         being thrown if the given String is equal to one of the
240  *         keywords above.
241  */
242     public static void checkIfNameIsKeyword(String value)
243     throws IllegalArgumentException {
244         for (String Keyword : SJavaSyntax.KEYWORDS) {
245             if (value.equals(Keyword)) {
246                 throw new IllegalArgumentException(
247                     "Parameter name is a keyword error");
248             }
249         }
250     }
251 }
252
253 /**
254  * Finding a parameter.
255  *
256  * @param currentScope
257  *         - the current scope.
258  * @return The found parameter. If it's a method calling then returning a
259  *         variable represents the return type of the method
260  * @throws NotExistVariableException
261  *         if the variable isn't exist.
262  * @throws NotMethodCallingException
263  *         if the method isn't exist.
264  * @throws CompilationException

```

```
264     */
265     public abstract Variable find(Scope currentScope, boolean creationCheck)
266         throws ParameterNotExistException;
267
268     public ArrayList<Parameter> getParameterList() {
269         return parameterList;
270     }
271
272     public String getName() {
273         return name;
274     }
275
276 }
```

19 oop/ex7/main/parameters/ParameterCompilationEx

```
1 package oop.ex7.main.parameters;
2
3 import oop.ex7.main.CompilationException;
4
5
6 /**
7  * This exception indicates that there is a compilation error in a parameter.
8  *
9  * @author roeial
10  *
11  */
12 public class ParameterCompilationException extends CompilationException {
13
14     public ParameterCompilationException(String errorMessage) {
15         super(errorMessage);
16     }
17
18     /**
19      *
20      */
21     private static final long serialVersionUID = 1L;
22
23 }
```

20 oop/ex7/main/parameters/ParameterNotExistException

```
1 package oop.ex7.main.parameters;
2
3 /**
4  * This exception indicates that a parameter isn't exist, a compilation error.
5  *
6  * @author roeial
7  *
8  */
9 public class ParameterNotExistException extends ParameterCompilationException {
10
11     public ParameterNotExistException(String errorMessage) {
12         super(errorMessage);
13     }
14
15     /**
16      *
17      */
18     private static final long serialVersionUID = 1L;
19
20 }
```

21 oop/ex7/main/scopes/ConditionalScope.java

```
1  package oop.ex7.main.scopes;
2
3  import oop.ex7.main.IllegalSyntaxException;
4  import oop.ex7.main.parameters.IllegalParameterException;
5  import oop.ex7.main.parameters.MethodCalling;
6  import oop.ex7.main.parameters.Parameter;
7  import oop.ex7.main.parameters.ParameterNotExistException;
8  import oop.ex7.main.parameters.variables.BooleanType;
9  import oop.ex7.main.parameters.variables.GenericType;
10 import oop.ex7.main.parameters.variables.Variable;
11
12 /**
13  * This class represents a scope containing a condition. (if / while scope).
14  * Extending the Scope class.
15  *
16  * @author roeial
17  *
18  */
19 public class ConditionalScope extends Scope {
20
21     private Parameter condition;
22
23     /**
24      * A data constructor.
25      *
26      * @param fatherScope
27      *      - the father scope of this scope.
28      * @param condition
29      *      - the condition of this scope.
30      * @throws IllegalParameterException
31      *      if the condition isn't match for any parameter type.
32      * @throws IllegalSyntaxException
33      *      if the condition is a method calling and there is a syntax
34      *      error.
35      */
36     public ConditionalScope(Scope fatherScope, String condition)
37         throws IllegalParameterException, IllegalSyntaxException {
38         super(fatherScope);
39         this.condition = Parameter.createOneParameter(condition);
40     }
41
42     /**
43      * Checking if the condition is a boolean parameter.
44      *
45      * @throws MismatchConditionTypeException
46      *      if the condition isn't a boolean parameter.
47      * @throws ParameterNotExistException
48      *      if the condition is a parameter and it doesn't exist.
49      */
50     public void CheckCondition() throws MismatchConditionTypeException,
51         ParameterNotExistException {
52         if (!(this.condition instanceof BooleanType)) {
53             if (this.condition instanceof GenericType
54                 || this.condition instanceof MethodCalling) {
55                 Variable foundVariable =
56                     this.condition.find(this.getFatherScope(), false);
57                 if (!(foundVariable instanceof BooleanType)) {
58                     throw new MismatchConditionTypeException(
59                         "Mismatch condition type error");
60                 }
61             }
62         }
63     }
64 }
```

```

60         }
61         if (this.condition instanceof GenericType
62             && !foundVariable.isInitialized()) {
63             throw new MismatchConditionTypeException(
64                 "Boolean variable not initialized in the condition");
65         }
66     } else {
67         throw new MismatchConditionTypeException(
68             "Mismatch condition type error");
69     }
70 }
71 }
72
73 public Parameter getCondition() {
74     return condition;
75 }
76 }

```

22 oop/ex7/main/scopes/MethodScope.java

```
1  package oop.ex7.main.scopes;
2
3  import java.util.regex.Matcher;
4  import java.util.regex.Pattern;
5
6  import oop.ex7.main.IllegalSyntaxException;
7  import oop.ex7.main.SJavaLine;
8  import oop.ex7.main.SJavaSyntax;
9  import oop.ex7.main.parameters.IllegalNameException;
10 import oop.ex7.main.parameters.Parameter;
11 import oop.ex7.main.parameters.variables.Variable;
12 import oop.ex7.main.parameters.variables.VariableFactory;
13
14
15 /**
16  * This class represents a method scope in the s-java file. Extends the Scope
17  * class.
18  *
19  * @author roeial
20  *
21  */
22 public class MethodScope extends Scope {
23
24     private final static String DECLARATION_PARAMETER_REGEX = "(?:\\s*"
25         + SJavaSyntax.TYPES_REGEX + "\\s*" + SJavaSyntax.METHOD_NAME_REGEX
26         + "\\s*)";
27     private final static int PARAMETER_TYPE_GROUP = 1;
28     private final static int PARAMETER_ARRAY_SIGN_GROUP = 2;
29     private final static int PARAMETER_NAME_GROUP = 3;
30     private String name;
31     private Variable returnType;
32     private int numOfParameters;
33
34     /**
35      * A data constructor.
36      *
37      * @param fatherScope
38      *      - the father scope of this method scope.
39      * @param returnType
40      *      - the return type of the method.
41      * @param arraySign
42      *      - the return type array sign.
43      * @param name
44      *      - the method name.
45      * @param parameters
46      *      - the declaration parameters of the method.
47      * @throws IllegalSyntaxException
48      *      if there is a syntax error in the method scope.
49      */
50     public MethodScope(Scope fatherScope, String returnType, String arraySign,
51         String name, String parameters) throws IllegalSyntaxException {
52         super(fatherScope);
53         Parameter.checkIfNameIsKeyword(name);
54         this.name = name;
55         if (returnType.equals("void")) {
56             this.returnType = null;
57         } else {
58             this.returnType =
59                 VariableFactory.createVariable(returnType, arraySign, null,
```

```

60         null);
61     }
62     /*
63     * Dividing the given parameters string, and creating each parameter
64     * while counting them.
65     */
66     this.numOfParameters = 0;
67     if (!parameters.equals("")) {
68         SJavaSyntax.checkIfStartOrEndWithComma(parameters);
69         String[] parametersArray = parameters.split(SJavaSyntax.COMMA);
70         Pattern parameterPattern =
71             Pattern.compile(DECLARATION_PARAMETER_REGEX);
72         Matcher parameterMatcher;
73         for (String currParameter : parametersArray) {
74             parameterMatcher = parameterPattern.matcher(currParameter);
75             if (parameterMatcher.matches()) {
76                 Variable declarationParameter =
77                     VariableFactory
78                         .createVariable(parameterMatcher
79                             .group(PARAMETER_TYPE_GROUP), parameterMatcher
80                             .group(PARAMETER_ARRAY_SIGN_GROUP),
81                             parameterMatcher.group(PARAMETER_NAME_GROUP),
82                             null);
83                 // Check if the variable name already exist
84                 // in the declaration
85                 for (SJavaLine currLine : this.getLineList()) {
86                     if (declarationParameter.getName().equals(
87                         ((Variable) currLine).getName())) {
88                         throw new IllegalArgumentException("
89                             + "Already exist variable error");
90                     }
91                 }
92                 // Making the declaration parameter initialized by
93                 // adding to it a raw parameter from the same type
94                 declarationParameter.getParameterList().add(
95                     VariableFactory.createVariable(parameterMatcher
96                         .group(PARAMETER_TYPE_GROUP), null, null, null));
97                 this.getLineList().add(declarationParameter);
98                 this.numOfParameters++;
99             } else {
100                 throw new IllegalArgumentException(
101                     "Illegal parameter of method declaration error");
102             }
103         }
104     }
105 }
106
107
108 public int getNumOfParameters() {
109     return numOfParameters;
110 }
111
112 public String getName() {
113     return name;
114 }
115
116 public Variable getReturnValue() {
117     return returnType;
118 }
119
120 }

```


23 oop/ex7/main/scopes/MismatchConditionTypeExce

```
1 package oop.ex7.main.scopes;
2
3 import oop.ex7.main.CompilationException;
4
5
6 /**
7  * This exception indicates that the condition is not a boolean type.
8  *
9  * @author roeial
10  *
11  */
12 public class MismatchConditionTypeException extends CompilationException {
13
14     public MismatchConditionTypeException(String errorMessage) {
15         super(errorMessage);
16     }
17
18     private static final long serialVersionUID = 1L;
19
20 }
```

24 oop/ex7/main/scopes/Scope.java

```
1  package oop.ex7.main.scopes;
2
3  import java.util.ArrayList;
4
5  import oop.ex7.main.SJavaLine;
6
7  /**
8   * This class represents a scope in the s-java file.
9   *
10  * @author roeial
11  *
12  */
13  public class Scope {
14
15      private ArrayList<SJavaLine> lineList;
16      private ArrayList<Scope> childScopeList;
17      private Scope fatherScope;
18
19      /**
20       * A data constructor.
21       *
22       * @param fatherScope
23       *      - The father scope of this scope.
24       */
25      public Scope(Scope fatherScope) {
26          this.fatherScope = fatherScope;
27          this.lineList = new ArrayList<SJavaLine>();
28          this.childScopeList = new ArrayList<Scope>();
29      }
30
31      public Scope getFatherScope() {
32          return fatherScope;
33      }
34
35      public ArrayList<SJavaLine> getLineList() {
36          return lineList;
37      }
38
39      public ArrayList<Scope> getChildScopeList() {
40          return childScopeList;
41      }
42  }
```

25 oop/ex7/main/parameters/variables/BooleanType.j

```
1 package oop.ex7.main.parameters.variables;
2
3 import oop.ex7.main.IllegalSyntaxException;
4 import oop.ex7.main.parameters.IllegalNameException;
5
6 /**
7  * This class is the BooleanType extends the variable class (true/false).
8  *
9  * @author roeial
10  *
11  */
12 public class BooleanType extends Variable {
13
14     /** The regular expression of a raw boolean */
15     public final static String BOOLEAN_REGEX = "\\s*true|false\\s*";
16
17     /**
18      * Constructing a raw boolean.
19      */
20     public BooleanType() {
21         super();
22     }
23
24     /**
25      * Constructing a return value boolean for a method.
26      *
27      * @param isArray
28      *      - represents if the return value boolean is an array.
29      */
30     public BooleanType(boolean isArray) {
31         super(isArray);
32     }
33
34     /**
35      * Constructing a new boolean variable.
36      *
37      * @param name
38      *      - the name of the boolean variable.
39      * @param isArray
40      *      - represents if the new boolean variable is an array.
41      * @throws IllegalNameException
42      *      if the boolean variable name isn't valid (a keyword).
43      */
44     public BooleanType(String name, boolean isArray)
45         throws IllegalNameException {
46         super(name, isArray);
47     }
48
49     /**
50      * Constructing a new boolean variable with an assignment value.
51      *
52      * @param name
53      *      - the name of the boolean variable.
54      * @param isArray
55      *      - represents if the new boolean variable is an array.
56      * @param assignmentValue
57      *      - the assignment value string.
58      * @throws IllegalNameException
59      *      if the boolean variable name isn't valid (a keyword).
```

```
60      * @throws IllegalSyntaxException
61      *           if there is a syntax error with the assignment string.
62      */
63      public BooleanType(String name, boolean isArray, String assignmentValue)
64          throws IllegalNameException, IllegalSyntaxException {
65          super(name, isArray, assignmentValue);
66      }
67 }
```

26 oop/ex7/main/parameters/variables/CharType.java

```
1 package oop.ex7.main.parameters.variables;
2
3 import oop.ex7.main.IllegalSyntaxException;
4 import oop.ex7.main.parameters.IllegalNameException;
5
6 /**
7  * This class is the CharType extends the variable class.
8  *
9  * @author roeial
10  *
11  */
12 public class CharType extends Variable {
13
14     /** The regular expression of a raw char */
15     public final static String CHAR_REGEX = "^\\s*\\'\\.?\\'\\s*$";
16
17     /**
18      * Constructing a raw char.
19      */
20     public CharType() {
21         super();
22     }
23
24     /**
25      * Constructing a return value char for a method.
26      *
27      * @param isArray
28      *      - represents if the return value char is an array.
29      */
30     public CharType(boolean isArray) {
31         super(isArray);
32     }
33
34     /**
35      * Constructing a new char variable.
36      *
37      * @param name
38      *      - the name of the char variable.
39      * @param isArray
40      *      - represents if the new char variable is an array.
41      * @throws IllegalNameException
42      *      if the char variable name isn't valid (a keyword).
43      */
44     public CharType(String name, boolean isArray) throws IllegalNameException {
45         super(name, isArray);
46     }
47
48     /**
49      * Constructing a new char variable with an assignment value.
50      *
51      * @param name
52      *      - the name of the char variable.
53      * @param isArray
54      *      - represents if the new char variable is an array.
55      * @param assignmentValue
56      *      - the assignment value string.
57      * @throws IllegalNameException
58      *      if the char variable name isn't valid (a keyword).
59      * @throws IllegalSyntaxException
```

```
60      *           if there is a syntax error with the assignment string.
61      */
62      public CharType(String name, boolean isArray, String assignmentValue)
63          throws IllegalArgumentException, IllegalSyntaxException {
64          super(name, isArray, assignmentValue);
65      }
66  }
```

27 oop/ex7/main/parameters/variables/DoubleType.java

```
1 package oop.ex7.main.parameters.variables;
2
3 import oop.ex7.main.IllegalSyntaxException;
4 import oop.ex7.main.parameters.IllegalNameException;
5 import oop.ex7.main.parameters.Operandable;
6
7 /**
8  * This class is the DoubleType extends the variable class (5,5.2), and it is
9  * also implements Operandable interface (5+3.2).
10  *
11  * @author roeial
12  *
13  */
14 public class DoubleType extends Variable implements Operandable {
15
16     /** The regular expression of a raw double */
17     public final static String DOUBLE_REGEX = "^\\s*\\|-?\\s*\\d+\\.\\.\\d+\\s*$";
18
19     /**
20      * Constructing a raw double.
21      */
22     public DoubleType() {
23         super();
24     }
25
26     /**
27      * Constructing a return value double for a method.
28      *
29      * @param isArray
30      *      - represents if the return value double is an array.
31      */
32     public DoubleType(boolean isArray) {
33         super(isArray);
34     }
35
36     /**
37      * Constructing a new double variable.
38      *
39      * @param name
40      *      - the name of the double variable.
41      * @param isArray
42      *      - represents if the new double variable is an array.
43      * @throws IllegalNameException
44      *      if the double variable name isn't valid (a keyword).
45      */
46     public DoubleType(String name, boolean isArray)
47         throws IllegalNameException {
48         super(name, isArray);
49     }
50
51     /**
52      * Constructing a new double variable with an assignment value.
53      *
54      * @param name
55      *      - the name of the double variable.
56      * @param isArray
57      *      - represents if the new double variable is an array.
58      * @param assignmentValue
59      *      - the assignment value string.
```

```
60      * @throws IllegalArgumentException
61      *         if the double variable name isn't valid (a keyword).
62      * @throws IllegalArgumentException
63      *         if there is a syntax error with the assignment string.
64      */
65     public DoubleType(String name, boolean isArray, String assignmentValue)
66         throws IllegalArgumentException, IllegalArgumentException {
67         super(name, isArray, assignmentValue);
68     }
69 }
```


28 oop/ex7/main/parameters/variables/GenericType.java

```
1 package oop.ex7.main.parameters.variables;
2
3 import oop.ex7.main.IllegalSyntaxException;
4 import oop.ex7.main.SJavaSyntax;
5 import oop.ex7.main.parameters.IllegalNameException;
6 import oop.ex7.main.parameters.IllegalParameterException;
7 import oop.ex7.main.parameters.Parameter;
8
9 /**
10  * This class is the GenericType variable, extends the variable class.
11  * GenericType is a variable which his type is unknown in the syntax check
12  * phase.
13  *
14  * @author roeial
15  *
16  */
17 public class GenericType extends Variable {
18
19     /** The regular expression of a generic variable */
20     public final static String GENERIC_REGEX = "^\\s*\\|-?\\s*"
21         + SJavaSyntax.VARIABLE_NAME_REGEX + "\\s*(?:"
22         + SJavaSyntax.ASSIGNMENT_ARRAY_SIGN + ")?\\s*$";
23     private Parameter arrayLocationParameter;
24
25     /**
26      * Constructing a new generic variable.
27      *
28      * @param name
29      *         - the name of the generic variable.
30      * @throws IllegalNameException
31      *         if the generic variable name isn't valid (a keyword).
32      */
33     public GenericType(String name) throws IllegalNameException {
34         super(name, false);
35     }
36
37     /**
38      * Constructing a new generic variable that's inside an array (abc[1]).
39      *
40      * @param name
41      *         - the name of the generic variable.
42      * @param arrayLocationParameter
43      *         - the location inside the array.
44      * @throws IllegalSyntaxException
45      *         if the array location parameter is a method calling with a
46      *         syntax error.
47      * @throws IllegalParameterException
48      *         if the array location parameter string isn't match for any
49      *         type.
50      * @throws IllegalArrayLocationException
51      *         if the array location is a raw integer that is negative.
52      */
53     public GenericType(String name, String arrayLocationParameter)
54         throws IllegalParameterException, IllegalSyntaxException,
55         IllegalArrayLocationException {
56         super(name, true);
57         this.arrayLocationParameter =
58             Parameter.createParameter(arrayLocationParameter);
59         // Checking if the location is a negative number
```

```

60         if (this.arrayLocationParameter instanceof IntegerType
61             && this.arrayLocationParameter.getParameterList().isEmpty()
62             && arrayLocationParameter.contains("-")) {
63             throw new IllegalArrayLocationException(
64                 "Illegal array location error");
65         }
66     }
67 }
68
69 public Parameter getArrayLocationParameter() {
70     return arrayLocationParameter;
71 }
72 }

```

29 oop/ex7/main/parameters/variables/IllegalArrayLoc

```
1 package oop.ex7.main.parameters.variables;
2
3 /**
4  * This exception indicates there is an error in the array location (abc[-1]).
5  *
6  * @author roeial
7  *
8  */
9 public class IllegalArrayLocationException extends
10     IllegalVariableSyntaxException {
11
12     public IllegalArrayLocationException(String errorMessage) {
13         super(errorMessage);
14     }
15
16     /**
17      *
18      */
19     private static final long serialVersionUID = 1L;
20
21 }
```

30 oop/ex7/main/parameters/variables/IllegalVariableS

```
1 package oop.ex7.main.parameters.variables;
2
3 import oop.ex7.main.parameters.IllegalParameterException;
4
5 /**
6  * This exception indicates that there is a syntax error in a variable.
7  *
8  * @author roeial
9  *
10 */
11 public class IllegalVariableSyntaxException extends IllegalParameterException {
12
13     public IllegalVariableSyntaxException(String errorMessage) {
14         super(errorMessage);
15     }
16
17     /**
18      *
19      */
20     private static final long serialVersionUID = 1L;
21
22 }
```

31 oop/ex7/main/parameters/variables/IllegalVariableType

```
1 package oop.ex7.main.parameters.variables;
2
3 import oop.ex7.main.IllegalSyntaxException;
4
5
6 /**
7  * This exception indicates that there is a syntax error in the variable type.
8  *
9  * @author roeial
10  *
11  */
12 public class IllegalVariableTypeException extends IllegalSyntaxException {
13
14     /**
15      * super the constructor with error message.
16      */
17     public IllegalVariableTypeException(String errorMessage) {
18         super(errorMessage);
19     }
20
21     private static final long serialVersionUID = 1L;
22
23 }
```

32 oop/ex7/main/parameters/variables/IntegerType.java

```
1 package oop.ex7.main.parameters.variables;
2
3 import oop.ex7.main.IllegalSyntaxException;
4 import oop.ex7.main.parameters.IllegalNameException;
5
6 /**
7  * This class represents the integer type variable, extends from double type.
8  *
9  * @author roeial
10  *
11  */
12 public class IntegerType extends DoubleType {
13
14     /** The regular expression of a raw integer */
15     public static final String INTEGER_REGEX = "^\\s*\\|-?\\s*\\d+\\s*$";
16
17     /**
18      * Constructing a raw integer.
19      */
20     public IntegerType() {
21         super();
22     }
23
24     /**
25      * Constructing a return value integer for a method.
26      *
27      * @param isArray
28      *      - represents if the return value integer is an array.
29      */
30     public IntegerType(boolean isArray) {
31         super(isArray);
32     }
33
34     /**
35      * Constructing a new integer variable.
36      *
37      * @param name
38      *      - the name of the integer variable.
39      * @param isArray
40      *      - represents if the new integer variable is an array.
41      * @throws IllegalNameException
42      *      if the integer variable name isn't valid (a keyword).
43      */
44     public IntegerType(String name, boolean isArray)
45         throws IllegalNameException {
46         super(name, isArray);
47     }
48
49     /**
50      * Constructing a new integer variable with an assignment value.
51      *
52      * @param name
53      *      - the name of the integer variable.
54      * @param isArray
55      *      - represents if the new integer variable is an array.
56      * @param assignmentValue
57      *      - the assignment value string.
58      * @throws IllegalNameException
59      *      if the integer variable name isn't valid (a keyword).
```

```
60      * @throws IllegalSyntaxException
61      *           if there is a syntax error with the assignment string.
62      */
63      public IntegerType(String name, boolean isArray, String assignmentValue)
64          throws IllegalNameException, IllegalSyntaxException {
65          super(name, isArray, assignmentValue);
66      }
67
68 }
```

33 oop/ex7/main/parameters/variables/MethodCalling

```
1 package oop.ex7.main.parameters.variables;
2
3 /**
4  * This exception indicates that there is a method calling return variable
5  * mismatch to the assigned variable.
6  *
7  * @author roeial
8  *
9  */
10 public class MethodCallingTypeMismatchException extends
11     VariableCompilationException {
12
13     public MethodCallingTypeMismatchException(String errorMessage) {
14         super(errorMessage);
15     }
16
17     /**
18      *
19      */
20     private static final long serialVersionUID = 1L;
21
22 }
```


34 oop/ex7/main/parameters/variables/NotOperandable

```
1 package oop.ex7.main.parameters.variables;
2
3 /**
4  * This exception indicates that a method calling return variable isn't
5  * operandable while using it with operand.
6  *
7  * @author roeial
8  *
9  */
10 public class NotOperandableMethodCallingException extends
11     VariableCompilationException {
12
13     public NotOperandableMethodCallingException(String errorMessage) {
14         super(errorMessage);
15     }
16
17     /**
18      *
19      */
20     private static final long serialVersionUID = 1L;
21
22 }
```

35 oop/ex7/main/parameters/variables/NotOperandable

```
1 package oop.ex7.main.parameters.variables;
2
3 /**
4  * This exception indicates that a variable isn't operandable while using it
5  * with operand.
6  *
7  * @author roeial
8  *
9  */
10 public class NotOperandableVariableException extends
11     VariableCompilationException {
12
13     public NotOperandableVariableException(String errorMessage) {
14         super(errorMessage);
15     }
16
17     /**
18      *
19      */
20     private static final long serialVersionUID = 1L;
21
22 }
```

36 oop/ex7/main/parameters/variables/StringType.java

```
1 package oop.ex7.main.parameters.variables;
2
3 import oop.ex7.main.IllegalSyntaxException;
4 import oop.ex7.main.parameters.IllegalNameException;
5
6 /**
7  * This class represents a string variable, extends from variable class.
8  *
9  * @author roeial
10  *
11  */
12 public class StringType extends Variable {
13
14     /** The regular expression of a raw String */
15     public final static String STRING_REGEX = "^\\s*\\.\\.\\s*$";
16
17     /**
18      * Constructing a raw String.
19      */
20     public StringType() {
21         super();
22     }
23
24     /**
25      * Constructing a return value String for a method.
26      *
27      * @param isArray
28      *      - represents if the return value String is an array.
29      */
30     public StringType(boolean isArray) {
31         super(isArray);
32     }
33
34     /**
35      * Constructing a new String variable.
36      *
37      * @param name
38      *      - the name of the String variable.
39      * @param isArray
40      *      - represents if the new String variable is an array.
41      * @throws IllegalNameException
42      *      if the String variable name isn't valid (a keyword).
43      */
44     public StringType(String name, boolean isArray)
45         throws IllegalNameException {
46         super(name, isArray);
47     }
48
49     /**
50      * Constructing a new String variable with an assignment value.
51      *
52      * @param name
53      *      - the name of the String variable.
54      * @param isArray
55      *      - represents if the new String variable is an array.
56      * @param assignmentValue
57      *      - the assignment value string.
58      * @throws IllegalNameException
59      *      if the String variable name isn't valid (a keyword).
```

```
60      * @throws IllegalSyntaxException
61      *           if there is a syntax error with the assignment string.
62      */
63      public StringType(String name, boolean isArray, String assignmentValue)
64          throws IllegalNameException, IllegalSyntaxException {
65          super(name, isArray, assignmentValue);
66      }
67 }
```

37 oop/ex7/main/parameters/variables/UninitializedVa

```
1 package oop.ex7.main.parameters.variables;
2
3 /**
4  * This exception indicates that an uninitialized variable used for assignment,
5  * compilation error.
6  *
7  * @author Roei
8  *
9  */
10 public class UninitializedVariableException extends
11     VariableCompilationException {
12
13     public UninitializedVariableException(String errorMessage) {
14         super(errorMessage);
15     }
16
17     /**
18      *
19      */
20     private static final long serialVersionUID = 1L;
21
22 }
```

38 oop/ex7/main/parameters/variables/Variable.java

```
1 package oop.ex7.main.parameters.variables;
2
3 import java.util.ArrayList;
4
5 import oop.ex7.main.AssignmentLine;
6 import oop.ex7.main.IllegalSyntaxException;
7 import oop.ex7.main.ReturnLine;
8 import oop.ex7.main.SJavaLine;
9 import oop.ex7.main.Sjavac;
10 import oop.ex7.main.parameters.IllegalNameException;
11 import oop.ex7.main.parameters.MethodCalling;
12 import oop.ex7.main.parameters.Operandable;
13 import oop.ex7.main.parameters.Parameter;
14 import oop.ex7.main.parameters.ParameterCompilationException;
15 import oop.ex7.main.parameters.ParameterNotExistException;
16 import oop.ex7.main.scopes.Scope;
17
18 /**
19  * This abstract class represent all of the legitimate variables in s-java. It
20  * extends the parameter class.
21  *
22  * @author roeial
23  *
24  */
25 public abstract class Variable extends Parameter {
26
27     private boolean isArray;
28
29     /**
30      * Constructing a raw variable.
31      */
32     protected Variable() {
33         super();
34     }
35
36     /**
37      * Constructing a new variable.
38      *
39      * @param name
40      *         - the name of the variable.
41      * @throws IllegalNameException
42      *         if the variable name isn't valid (a keyword).
43      */
44     protected Variable(String name) throws IllegalNameException {
45         super(name);
46     }
47
48     /**
49      * Constructing a new variable.
50      *
51      * @param isArray
52      *         - represents if the variable is an array.
53      */
54     protected Variable(boolean isArray) {
55         this.isArray = isArray;
56     }
57
58     /**
59      * Constructing a new variable.
```

```

60     *
61     * @param name
62     *     - the name of the variable.
63     * @param isArray
64     *     - represents if the variable is an array.
65     * @throws IllegalArgumentException
66     *     if the variable name isn't valid (a keyword).
67     */
68     protected Variable(String name, boolean isArray)
69     throws IllegalArgumentException {
70         super(name);
71         this.isArray = isArray;
72     }
73
74     /**
75     * Constructing a new variable with an assignment value.
76     *
77     * @param name
78     *     - the name of the variable.
79     * @param isArray
80     *     - represents if the variable is an array.
81     * @param assignmentValue
82     *     - the assignment value string.
83     * @throws IllegalArgumentException
84     *     if there is a syntax error with the assignment string.
85     */
86     protected Variable(String name, boolean isArray, String assignmentValue)
87     throws IllegalArgumentException {
88         super(name);
89         this.isArray = isArray;
90         if (isArray) {
91             this.parameterList = arrayAssignment(assignmentValue);
92         } else {
93             this.parameterList.add(createParameter(assignmentValue));
94         }
95     }
96
97     public void isValid(Scope currentScope)
98     throws VariableAlreadyExistException, ParameterCompilationException {
99         try {
100             this.find(currentScope, true);
101             throw new VariableAlreadyExistException(
102                 "Already exist variable error");
103         } catch (VariableNotExistException e) {
104             for (Parameter currParameter : this.getParameterList()) {
105                 this.checkAssignment(currParameter, currentScope);
106             }
107         }
108     }
109
110     /**
111     * Checking if the assignment of the parameter input is valid for this
112     * variable.
113     *
114     * @param parameter
115     *     - the assigning parameter.
116     * @param currentScope
117     *     - the current scope of the variable.
118     * @throws ParameterNotExistException
119     *     if the parameter doesn't exist.
120     * @throws NotOperandableVariableException
121     *     if assigning parameters with operand and one of them is a
122     *     variable that isn't operandable.
123     * @throws VariableTypeMismatchException
124     *     if assigning a variable that isn't from the same type as the
125     *     assigned one.
126     * @throws UninitializedVariableException
127     *     if assigning a variable that isn't initialized.

```

```

128     * @throws VoidMethodException
129     *         if assigning a method that returns void.
130     * @throws NotOperableMethodCallingException
131     *         if assigning parameters with operand and one of them is a
132     *         method calling that returns a variable that isn't
133     *         operable.
134     * @throws MethodCallingTypeMismatchException
135     *         if assigning a method calling that returns a variable that
136     *         isn't from the same type as the assigned one.
137     * @throws ParameterCompilationException
138     *         if the parameter is a method calling and there is a
139     *         compilation error when checking validation.
140     */
141     public void checkAssignment(Parameter parameter, Scope currentScope)
142         throws ParameterNotExistException, NotOperableVariableException,
143         VariableTypeMismatchException, UninitializedVariableException,
144         VoidMethodException, NotOperableMethodCallingException,
145         MethodCallingTypeMismatchException, ParameterCompilationException {
146         ArrayList<Parameter> parameters = new ArrayList<Parameter>();
147         Variable foundVariable;
148         parameters.add(parameter);
149         // Checking if the assignment is with two parameters with operand
150         if (parameter instanceof MethodCalling) {
151             if (parameter.getParameterList().size() >
152                 ((MethodCalling) parameter).getNumOfParameters()) {
153                 parameters.add(parameter.getParameterList().get(
154                     ((MethodCalling) parameter).getNumOfParameters()));
155             }
156         } else if (!parameter.getParameterList().isEmpty()) {
157             parameters.add(parameter.getParameterList().get(0));
158         }
159         for (Parameter currParameter : parameters) {
160             if (currParameter instanceof GenericType) {
161                 foundVariable = currParameter.find(currentScope, false);
162                 if (parameters.size() == 2
163                     && !(foundVariable instanceof Operable)) {
164                     throw new NotOperableVariableException(
165                         "Assignment using operand with a parameter that"
166                         + " is not operable");
167                 }
168                 if (!this.getClass().isInstance(foundVariable)) {
169                     throw new VariableTypeMismatchException(
170                         "Assignment of a variable not from same type");
171                 }
172                 if (!foundVariable.isInitialized()) {
173                     throw new UninitializedVariableException(
174                         "Assignment of an uninitialized variable");
175                 }
176             } else if (currParameter instanceof MethodCalling) {
177                 ((MethodCalling) currParameter).isValid(currentScope);
178                 foundVariable = currParameter.find(currentScope, false);
179                 if (foundVariable == null) {
180                     throw new VoidMethodException(
181                         "Assignment using a method calling that return void");
182                 }
183                 if (parameters.size() == 2
184                     && !(foundVariable instanceof Operable)) {
185                     throw new NotOperableMethodCallingException(
186                         "Assignment using operand with a parameter that is"
187                         + " not operable");
188                 }
189                 if (!this.getClass().isInstance(foundVariable)) {
190                     throw new MethodCallingTypeMismatchException(
191                         "Assignment of a method calling that not returning "
192                         + "same type");
193                 }
194             } else if (parameters.size() == 2
195                 && !(this instanceof Operable)) {

```



```

196         throw new NotOperandableVariableException(
197             "Assignment using operand with a parameter that is not "
198             + "operanable");
199     } else if (!this.getClass().isInstance(currParameter)) {
200         throw new VariableTypeMismatchException(
201             "Assignment of a raw variable that not from the same "
202             + "type");
203     }
204 }
205 }
206
207 public Variable find(Scope currentScope, boolean creationCheck)
208     throws VariableNotExistException {
209     for (SJavaLine currLine : currentScope.getLineList()) {
210         // check for all the cases which there are not the return line
211         if (!(currLine instanceof ReturnLine)) {
212             // check if the variable exists
213             if (this == currLine
214                 || (currLine instanceof Variable && ((Variable) currLine)
215                     .getParameterList().contains(this))
216                 || (currLine instanceof AssignmentLine &&
217                     ((AssignmentLine) currLine)
218                     .getAssignedGeneric() == this)) {
219                 break;
220             }
221             if (currLine instanceof Variable
222                 && ((Variable) currLine).getName().equals(this.getName())) {
223                 return (Variable) currLine;
224             }
225         }
226     }
227     // going to the father scope.
228     currentScope = currentScope.getFatherScope();
229     if (creationCheck) {
230         if (currentScope != null && currentScope != Sjavac.MAIN_SCOPE) {
231             return (Variable) recursiveFind(currentScope, creationCheck);
232         }
233     } else {
234         if (currentScope != null) {
235             return (Variable) recursiveFind(currentScope, creationCheck);
236         }
237     }
238     throw new VariableNotExistException("Variable doesn't exist");
239 }
240
241 private Parameter recursiveFind(Scope currentScope, boolean creationCheck)
242     throws VariableNotExistException {
243     // running on the lines of current scope.
244     for (SJavaLine currLine : currentScope.getLineList()) {
245         // if found it than return the current line as variable or
246         // parameter depends on the instance it has.
247         if (currLine instanceof Variable
248             && ((Variable) currLine).getName().equals(this.getName())) {
249             return (Variable) currLine;
250         }
251     }
252     // going to the father scope.
253     currentScope = currentScope.getFatherScope();
254     if (creationCheck) {
255         if (currentScope != Sjavac.MAIN_SCOPE) {
256             return (Variable) recursiveFind(currentScope, creationCheck);
257         }
258     } else {
259         if (currentScope != null) {
260             return (Variable) recursiveFind(currentScope, creationCheck);
261         }
262     }
263     throw new VariableNotExistException("Variable doesn't exist");

```

```
264     }
265
266     /**
267      * Checking if a variable is initialized.
268      *
269      * @return True if the variable initialized, false otherwise.
270      */
271     public boolean isInitialized() {
272         return !this.getParameterList().isEmpty();
273     }
274
275     public boolean isArray() {
276         return isArray;
277     }
278 }
```

39 oop/ex7/main/parameters/variables/VariableAlready

```
1 package oop.ex7.main.parameters.variables;
2
3 /**
4  * This exception extends the CompilationException and being thrown when a
5  * variable is already exist.
6  *
7  * @author roeial
8  *
9  */
10 public class VariableAlreadyExistException extends
11     VariableCompilationException {
12
13     /**
14      * super the constructor with error message.
15      *
16      * @param errorMessage
17      */
18     public VariableAlreadyExistException(String errorMessage) {
19         super(errorMessage);
20     }
21
22     private static final long serialVersionUID = 1L;
23
24 }
```

40 oop/ex7/main/parameters/variables/VariableCompi

```
1 package oop.ex7.main.parameters.variables;
2
3 import oop.ex7.main.parameters.ParameterCompilationException;
4
5 /**
6  * This exception indicates that there is a compilation error with the variable.
7  *
8  * @author roeial
9  *
10 */
11 public class VariableCompilationException extends
12     ParameterCompilationException {
13
14     public VariableCompilationException(String errorMessage) {
15         super(errorMessage);
16     }
17
18     /**
19      *
20      */
21     private static final long serialVersionUID = 1L;
22
23 }
```

41 oop/ex7/main/parameters/variables/VariableFactor

```
1 package oop.ex7.main.parameters.variables;
2
3 import oop.ex7.main.IllegalSyntaxException;
4 import oop.ex7.main.parameters.IllegalNameException;
5
6 /**
7  * This class represents the variable factory.
8  *
9  * @author roeial
10  *
11  */
12 public class VariableFactory {
13
14     /**
15      * Creating a variable.
16      *
17      * @param type
18      *      - the variable type.
19      * @param arraySign
20      *      - the array sign.
21      * @param name
22      *      - the name of the variable.
23      * @param assignmentvalue
24      *      - the assignment value of the new variable.
25      * @return The new variable.
26      * @throws IllegalNameException
27      *      if the name of the variable is a keyword.
28      * @throws IllegalSyntaxException
29      *      if the assignment string has a syntax error.
30      */
31     public static Variable createVariable(String type, String arraySign,
32     String name, String assignmentvalue) throws IllegalNameException,
33     IllegalSyntaxException {
34         boolean isArray = false;
35         // Checking if the variable is array
36         if (arraySign != null) {
37             isArray = !(arraySign.equals(""));
38         }
39         Variable newVariable;
40         // Creating a variable parameter
41         switch (type) {
42             case "int":
43                 if (name == null) {
44                     if (arraySign == null) {
45                         // Creating a raw parameter
46                         newVariable = new IntegerType();
47                     } else {
48                         // Creating a return type variable for a method declaration
49                         newVariable = new IntegerType(isArray);
50                     }
51                 } else if (assignmentvalue != null) {
52                     newVariable = new IntegerType(name, isArray, assignmentvalue);
53                 } else {
54                     newVariable = new IntegerType(name, isArray);
55                 }
56                 break;
57             case "boolean":
58                 if (name == null) {
59                     if (arraySign == null) {
```

```

60         // Creating a raw parameter
61         newVariable = new BooleanType();
62     } else {
63         // Creating a return type variable for a method declaration
64         newVariable = new BooleanType(isArray);
65     }
66 } else if (assignmentvalue != null) {
67     newVariable = new BooleanType(name, isArray, assignmentvalue);
68 } else {
69     newVariable = new BooleanType(name, isArray);
70 }
71 break;
72 case "String":
73     if (name == null) {
74         if (arraySign == null) {
75             // Creating a raw parameter
76             newVariable = new StringType();
77         } else {
78             // Creating a return type variable for a method declaration
79             newVariable = new StringType(isArray);
80         }
81     } else if (assignmentvalue != null) {
82         newVariable = new StringType(name, isArray, assignmentvalue);
83     } else {
84         newVariable = new StringType(name, isArray);
85     }
86     break;
87 case "char":
88     if (name == null) {
89         if (arraySign == null) {
90             // Creating a raw parameter
91             newVariable = new CharType();
92         } else {
93             // Creating a return type variable for a method declaration
94             newVariable = new CharType(isArray);
95         }
96     } else if (assignmentvalue != null) {
97         newVariable = new CharType(name, isArray, assignmentvalue);
98     } else {
99         newVariable = new CharType(name, isArray);
100     }
101     break;
102 case "double":
103     if (name == null) {
104         if (arraySign == null) {
105             // Creating a raw parameter
106             newVariable = new DoubleType();
107         } else {
108             // Creating a return type variable for a method declaration
109             newVariable = new DoubleType(isArray);
110         }
111     } else if (assignmentvalue != null) {
112         newVariable = new DoubleType(name, isArray, assignmentvalue);
113     } else {
114         newVariable = new DoubleType(name, isArray);
115     }
116     break;
117 default:
118     throw new IllegalVariableTypeException("Variable type error");
119 }
120 return newVariable;
121 }
122 }

```

42 oop/ex7/main/parameters/variables/VariableNotEx

```
1 package oop.ex7.main.parameters.variables;
2
3 import oop.ex7.main.parameters.ParameterNotExistException;
4
5 /**
6  * This exception indicates that a variable isn't exist, compilation error.
7  *
8  * @author roeial
9  *
10 */
11 public class VariableNotExistException extends ParameterNotExistException {
12
13     public VariableNotExistException(String errorMessage) {
14         super(errorMessage);
15     }
16
17     /**
18      *
19      */
20     private static final long serialVersionUID = 1L;
21
22 }
```

43 oop/ex7/main/parameters/variables/VariableTypeMismatchException

```
1 package oop.ex7.main.parameters.variables;
2
3 /**
4  * This exception indicates that there is a variable mismatch to the assigned
5  * variable.
6  *
7  * @author roeial
8  *
9  */
10 public class VariableTypeMismatchException extends
11     VariableCompilationException {
12
13     public VariableTypeMismatchException(String errorMessage) {
14         super(errorMessage);
15     }
16
17     /**
18      *
19      */
20     private static final long serialVersionUID = 1L;
21
22 }
```


44 oop/ex7/main/parameters/variables/VoidMethodEx

```
1 package oop.ex7.main.parameters.variables;
2
3 /**
4  * This exception indicates that there is a method calling returning void used
5  * in assignment.
6  *
7  * @author roeial
8  *
9  */
10 public class VoidMethodException extends VariableCompilationException {
11
12     public VoidMethodException(String errorMessage) {
13         super(errorMessage);
14     }
15
16     /**
17      *
18      */
19     private static final long serialVersionUID = 1L;
20
21 }
```