Contents

1 Basic Test Results

```
Logins: roeia1



compiling with
    javac -cp .:/cs/course/2013/oop/lib/junit4.jar  *.java oop/ex7/main/*.java


tests output :
        Perfect!
```

2 README

```
roeia1
yinnonbar


File Description
----------------

    The file consists 4 packages :

    1. main - which contains :

    AssignmentLine - This class represents an s-java assignment line. (X = Y)

    CompilationException - An exception for compilation errors.

    IllegalReturnLocationUsageException - This exception extends the
    CompilationException and being thrown when an illegal value is given as
    the return location.

    IllegalSyntaxException - This exception extends the Exception class and
    being thrown when an illegal syntax line is given.

    Parser - This class is the parser. It is reading the lines in the file
    and turning it to objects while checking each line syntax.

    ReturnLine - This class represents a return line
    (Consist the "return" expression). Extending the SjavaLine class.

    Sjavac - This class is the main class of program. This is the compiler, it
    is taking the lines from the parser and prints the matching message.

    SJavaLine - This class represents an s-java line in the file being read.

    SJavaSyntax - This class represents all the valid syntax for a line in
    s-java file.

    2. main.parameters - which contains :

    IllegalNameException - This exception indicates that a name is a keyword.

    IllegalParameterException - This exception indicates that there is a
    parameter syntax exception.

    MethodCalling - This class represents a s-java line method calling.
    Extending the Parameter class.

    MethodCallingException - This exception indicates that there is a
    compilation error in a method calling.

    MethodCallingNotExistException - This exception indicates that the method
    calling isn't exist, a compilation error.

    Operandable - This interface indicates that a parameter is operandable.

    Parameter - This abstract class is the parameter class, extends SJavaLine
    and designed to work with initializing variables and method calling.

    ParameterCompilationException - This exception indicates that there is a
    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 :

    ConditionalScope - This class represents a scope containing a condition.
    (if / while scope). Extending the Scope class.

    MethodScope - This class represents a method scope in the s-java file.
    Extends the Scope class.

    MismatchConditionTypeException - This exception indicates that the
    condition is not a boolean type.

    Scope - This class represents a scope in the s-java file.

    README - This file.

Design
------

    In this project, since we have been asked to build a compiler,
    there is a total separation between the
    syntax checking and the compilation checking (the "IsValid" method).
    Basically, we created a tree when each node is a scope.
    There are 2 different scopes that inheriting from the scope class,
    the method scope and the simple scope (an if/while scope).
    Each scope also have an array list of SJavaLine object,
    this line represent everything but a scope.
    First, in the syntax process we created a switch for each line regex
    we could encounter when reading the lines from the file.
    This switch knows what regex the line match with an enum that check it.
    (this enum is in the SJavaSyntax class that also contains most of the
    complier regexes)
    Once we know what regex this line match to, we created an SJavaLine object
    and added it to the relevant scope SJavaLine array.
    After all the lines in the file converted to objects and the syntax
    process is complete, we checked each SJavaLine with an abstract method
    in the SJavaLine abstract class "IsValid".(each line option class inherit
    from the abstract class SJavaLine)
    We separated the packages in this project inside the main package
    "filescript" to a scope package and an SJavaLine package.
    We chose to used exceptions in the error handling issue because of the fact
    that in this way we can use hierarchy in the exception catching.
    Since we are using different exception classes with inheritance,
    when an exception being catch we can throw a precise error message
    indicating the mistake. (we also implemented the print stack trace in each
    exception with the constructor of the message)
    In case we need to add a new type of variable we just need to implement it
    while extending from the Variable abstract class.(Obviously we will need
    to add the new type name to the SJavaSyntax different regexes).
    Implementing an if-else block would be quite easy.
    Since we made a simple block class containing a condition representing
    both if and while blocks, we will need to separate them to different
    classes.
    When we encounter an else scope we will create an object of a Scope type
    and the syntax check would be checking in the father array of child scopes
    if before the this else scope there is an instance of an if scope.
    We used variable declaration regex and declaration method regex with
    groups to get all the data required for the constructor of each class.
    In variable declaration regex the groups are the following: type, name,
    equal sign(doesn’t have to appear), value(doesn’t have to appear).
    In declaration method regex the groups are the following: return type,
    name, all the parameters(doesn’t have to appear).
    Additionally, we did an inheritance from integer to double because that
    in the assignment integer can be assigned to double but not the other way
    around. We also did an Operandable interface indicating if a parameter
    can be with operand or not.
    Because of this separation from parsing to compilation in this project,
    We really think the design and implementation stands out from the rest.
    (Did I hear a bonus? :P)
```

3 oop/ex7/main/AssignmentLine.java

```java
package oop.ex7.main;

import java.util.ArrayList;

import oop.ex7.main.parameters.Parameter;
import oop.ex7.main.parameters.ParameterCompilationException;
import oop.ex7.main.parameters.variables.GenericType;
import oop.ex7.main.parameters.variables.IntegerType;
import oop.ex7.main.parameters.variables.Variable;
import oop.ex7.main.scopes.Scope;

/**
 * This class represents an s-java assignment line. (X = Y)
 *
 * @author roeia1
 *
 */
public class AssignmentLine extends SJavaLine {

    private GenericType assignedGeneric;
    private Parameter assigningParameter;

    /**
     * A data constructor.
     *
     * @param assignedVariable
     *            - the name of the variable being assigned.
     * @param assignedValue
     *            - the assignment value.
     * @throws IllegalSyntaxException
     *             if there is a syntax error in the assignment line.
     */
    public AssignmentLine(String assignedVariable, String assignedValue)
        throws IllegalSyntaxException {
        Parameter assignedParameter =
            Parameter.createOneParameter(assignedVariable);
        if (assignedParameter instanceof GenericType) {
            this.assignedGeneric = (GenericType) assignedParameter;
        } else {
            throw new IllegalSyntaxException(
                "Illegal assignment line syntax error");
        }
        assigningParameter = Parameter.createParameter(assignedValue);
    }

    public void isValid(Scope currentScope)
        throws ParameterCompilationException {
        // Checking the array location if it is array
        if (assignedGeneric.isArray()) {
            ArrayList<Parameter> arrayLocationParameters =
                new ArrayList<Parameter>();
            // Creating an array from the parameters of the array location
            arrayLocationParameters.add(assignedGeneric
                .getArrayLocationParameter());
            if (!assignedGeneric.getArrayLocationParameter()
                .getParameterList().isEmpty()) {
                arrayLocationParameters.add(assignedGeneric
                    .getArrayLocationParameter().getParameterList().get(0));
            }
            for (Parameter currArrayLocationParameter :
                arrayLocationParameters) {
                // If the parameter is a generic or raw and not being integer,
                // or a method calling not returning integer throw exception
                new IntegerType().checkAssignment(currArrayLocationParameter,
                    currentScope);
            }
        }
        // Finding the assigned generic variable
        Variable assignedVariable =
            this.assignedGeneric.find(currentScope, false);
        // Checking if its a valid assignment
        assignedVariable
            .checkAssignment(this.assigningParameter, currentScope);

    }

    public Variable getAssignedGeneric() {
        return assignedGeneric;
    }

    public Parameter getAssigningParameter() {
        return assigningParameter;
    }
}
```

4 oop/ex7/main/CompilationException.java

```java
package oop.ex7.main;

/**
 * An exception for compilation errors.
 *
 * @author roeia1
 *
 */
public class CompilationException extends Exception {

    /**
     * super the constructor with error message.
     */
    public CompilationException(String errorMessage) {
        super(errorMessage);
    }

    private static final long serialVersionUID = 1L;

}
```

5 oop/ex7/main/IllegalReturnLocationUsageException

```java
package oop.ex7.main;

/**
 * This exception extends the CompilationException and being thrown when an
 * illegal value is given as the return location.
 *
 * @author roeia1
 *
 */
public class IllegalReturnLocationUsageException extends CompilationException {

    /**
     * super the constructor with error message.
     */
    public IllegalReturnLocationUsageException(String errorMessage) {
        super(errorMessage);
    }

    private static final long serialVersionUID = 1L;

}
```

6 oop/ex7/main/IllegalSyntaxException.java

```java
package oop.ex7.main;

/**
 * This exception extends the Exception class and being thrown when an illegal
 * syntax line is given.
 *
 * @author roeia1
 *
 */
public class IllegalSyntaxException extends Exception {

    /**
     * super the constructor with error message.
     */
    public IllegalSyntaxException(String errorMessage) {
        super(errorMessage);
    }

    private static final long serialVersionUID = 1L;

}
```

7 oop/ex7/main/Parser.java

```java
package oop.ex7.main;

import java.io.FileReader;
import java.io.IOException;
import java.io.LineNumberReader;

import oop.ex7.main.parameters.MethodCalling;
import oop.ex7.main.parameters.variables.VariableFactory;
import oop.ex7.main.scopes.ConditionalScope;
import oop.ex7.main.scopes.MethodScope;
import oop.ex7.main.scopes.Scope;

/**
 * This class is the parser. It is reading the lines in the file and turning it
 * to objects while checking each line syntax.
 *
 * @author roeia1
 *
 */
public class Parser {
    /**
     * Parsing the file with the given file location, checking for valid s-java
     * syntax.
     *
     * @param fileLocation
     *            - the file location.
     * @throws IllegalSyntaxException
     *             if there is a syntax error.
     */
    public static void Parse(String fileLocation)
        throws IllegalSyntaxException, IOException {
        LineNumberReader lineReader =
            new LineNumberReader(new FileReader(fileLocation));
        Scope currentScope = Sjavac.MAIN_SCOPE;
        String currLine = lineReader.readLine();
        // reading the given file while the current line is not null, means
        // there's a line to read, dividing it to cases with switch
        // according to the matching regex
        while (currLine != null) {
            SJavaSyntax.Syntax syntax = SJavaSyntax.getSyntax(currLine);
            switch (syntax) {
            case BLANK_LINE:
                break;
            case COMMENT:
                break;
            case END_OF_SCOPE:
                if (currentScope == null) {
                    throw new IllegalSyntaxException("Illegal } usage error");
                } else {
                    currentScope = currentScope.getFatherScope();
                }
                break;
            case DECLARATION_METHOD:
                // Method calling
                if (syntax.getLineMatcher().group(
                    SJavaSyntax.METHOD_RETURN_TYPE_GROUP) == null) {
                    // Checking if called in main scope and throw exception
                    // if it does since it's an illegal calling for method
                    // at the main scope.
                    if (currentScope == Sjavac.MAIN_SCOPE) {
                        throw new IllegalSyntaxException(
                            "Method calling outside a method "
                                + "without assignment to a member");
                    } else {
                        currentScope.getLineList().add(
                            new MethodCalling(syntax.getLineMatcher().group(
                                SJavaSyntax.METHOD_NAME_GROUP), syntax
                                .getLineMatcher().group(
                                    SJavaSyntax.METHOD_PARAMETERS_GROUP)));
                    }
                    // Method declaration
                } else {
                    // Checking if declared in main scope
                    if (currentScope == Sjavac.MAIN_SCOPE) {
                        Sjavac.MAIN_SCOPE
                            .getChildScopeList()
                            .add(
                                new MethodScope(
                                    Sjavac.MAIN_SCOPE,
                                    syntax.getLineMatcher().group(
                                        SJavaSyntax.METHOD_RETURN_TYPE_GROUP),
                                    syntax
                                        .getLineMatcher()
                                        .group(
                                            SJavaSyntax
                                            .METHOD_RETURN_ARRAY_SIGN_GROUP),
                                    syntax.getLineMatcher().group(
                                        SJavaSyntax.METHOD_NAME_GROUP),
                                    syntax.getLineMatcher().group(
                                        SJavaSyntax.METHOD_PARAMETERS_GROUP)));
                        currentScope =
                            currentScope.getChildScopeList().get(
                                currentScope.getChildScopeList().size() - 1);
                    } else {
                        throw new IllegalSyntaxException(
                            "Method declaration not in main scope error");
                    }
                }
                break;
            case RETURN:
                currentScope.getLineList().add(
                    new ReturnLine(syntax.getLineMatcher().group(
                        SJavaSyntax.RETURN_VALUE_GROUP)));
                break;
            case SIMPLE_SCOPE:
                // Checking if the simple scope opens in the main scope
                // (if / while usage in main scope)
                if (currentScope == Sjavac.MAIN_SCOPE) {
                    throw new IllegalSyntaxException(
                        "if/while usage out of a method scope error");
                    // else adding that simple scope as a child to the
                    // current
                    // scope.
                } else {
                    currentScope.getChildScopeList().add(
                        new ConditionalScope(currentScope, syntax
                            .getLineMatcher().group(
                                SJavaSyntax.SIMPLE_SCOPE_CONDITION_GROUP)));
                    currentScope =
                        currentScope.getChildScopeList().get(
                            currentScope.getChildScopeList().size() - 1);
                }
                break;
            case VARIABLE:
                // in case the line matching to a variable send it to the
                // variable factory.
                currentScope.getLineList().add(
                    VariableFactory.createVariable(syntax.getLineMatcher()
                        .group(SJavaSyntax.VARIABLE_TYPE_GROUP), syntax
                        .getLineMatcher().group(
                            SJavaSyntax.VARIABLE_ARRAY_SIGN_GROUP), syntax
                        .getLineMatcher().group(
                            SJavaSyntax.VARIABLE_NAME_GROUP), syntax
                        .getLineMatcher().group(
                            SJavaSyntax.VARIABLE_VALUE_GROUP)));
                break;
            case ASSIGNMENT:
                // in case of an assignment (=) sending the current line to
                // assignment line constructor.
                currentScope.getLineList().add(
                    new AssignmentLine(syntax.getLineMatcher().group(
                        SJavaSyntax.ASSIGNMENT_VARIABLE_GROUP), syntax
                        .getLineMatcher().group(
                            SJavaSyntax.ASSIGNMENT_VALUE_GROUP)));
                break;
            case CALLING_METHOD:
                currentScope.getLineList().add(
                    new MethodCalling(syntax.getLineMatcher().group(
                        SJavaSyntax.METHOD_CALLING_NAME_GROUP), syntax
                        .getLineMatcher().group(
                            SJavaSyntax.METHOD_CALLING_PARAMETERS_GROUP)));
                break;
            default:
                // in case that the line is not matching to any of the
                // cases above throw an illegal syntax exception.
                throw new IllegalSyntaxException("Illegal syntax line error");

            }
            currLine = lineReader.readLine();
        }
        lineReader.close();
    }
}
```

8 oop/ex7/main/ReturnLine.java

```java
package oop.ex7.main;

import java.util.ArrayList;

import oop.ex7.main.parameters.IllegalParameterException;
import oop.ex7.main.parameters.Parameter;
import oop.ex7.main.parameters.variables.Variable;
import oop.ex7.main.scopes.MethodScope;
import oop.ex7.main.scopes.Scope;

/**
 * This class represents a return line (Consist the "return" expression).
 * Extending the SjavaLine class.
 *
 * @author roeia1
 *
 */
public class ReturnLine extends SJavaLine {

    private ArrayList<Parameter> parameterList;
    private boolean isRawArray;

    /**
     * A data constructor.
     *
     * @param returnValue
     *             - the return value.
     * @throws IllegalParameterException
     *              if one of the parameters string isn't match for any type.
     * @throws IllegalSyntaxException
     *              if there is a syntax error in the parameters string.
     */
    public ReturnLine(String returnValue) throws IllegalParameterException,
        IllegalSyntaxException {
        isRawArray = false;
        parameterList = new ArrayList<Parameter>();
        if (!returnValue.equals("")) {
            // Check if the return value is an array
            if (returnValue.matches(SJavaSyntax.ARRAY_ASSIGNMENT_REGEX)) {
                parameterList = Parameter.arrayAssignment(returnValue);
                isRawArray = true;
            } else {
                parameterList.add(Parameter.createParameter(returnValue));
            }
        }
    }

    public void isValid(Scope currentScope) throws CompilationException {
        if (currentScope instanceof MethodScope) {
            // Checking if the return line in the method scope is not the last
            // line
            if (currentScope.getLineList().indexOf(this) != currentScope
                .getLineList().size() - 1) {
                throw new IllegalReturnLocationUsageException(
                    "Illegal return location usage inside a method scope");
            }
        } else {
            currentScope = currentScope.getFatherScope();
            while (!(currentScope instanceof MethodScope)) {
                currentScope = currentScope.getFatherScope();
            }
        }
        Variable methodReturnValue =
            ((MethodScope) currentScope).getReturnValue();
        // Return line inside a void return_type method
        if (methodReturnValue == null) {
            if (!this.parameterList.isEmpty()) {
                throw new CompilationException(
                    "Return a value in a void return_type method");
            }
        } else {
            if (methodReturnValue.isArray()) {
                if (this.parameterList.isEmpty() && !this.isRawArray) {
                    throw new CompilationException(
                        "Mismatch type between the return value and line, "
                            + "one is array and one isn't");
                }
            } else {
                if (this.isRawArray) {
                    throw new CompilationException(
                        "Returning raw array when the return "
                            + "value of the method isn't array");
                }
                if (this.parameterList.isEmpty()) {
                    throw new CompilationException(
                        "No return parameter in a method "
                            + "that doesn't return void");
                }
            }
            // Checking if every parameter in the return line matching the
            // return type of the method
            for (Parameter currParameter : this.parameterList) {
                methodReturnValue.checkAssignment(currParameter, currentScope);
            }
        }
    }

    public ArrayList<Parameter> getParameterList() {
        return parameterList;
    }

    public boolean isRawArray() {
        return isRawArray;
    }
}
```

9 oop/ex7/main/SJavaLine.java

```java
package oop.ex7.main;

import oop.ex7.main.scopes.Scope;

/**
 * This class represents an s-java line in the file being read.
 *
 * @author roeia1
 *
 */
public abstract class SJavaLine {

    /**
     * This method check if the current s-java line is compilation valid.
     *
     * @param currentScope
     *            - the current scope.
     * @throws CompilationException
     *             if the s-java line has a compilation error.
     */
    public abstract void isValid(Scope currentScope)
        throws CompilationException;
}
```

10 oop/ex7/main/SJavaSyntax.java

```java
package oop.ex7.main;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * This class represents all the valid syntax for a line in s-java file.
 *
 * @author roeia1
 *
 */
public class SJavaSyntax {

    public final static int RETURN_VALUE_GROUP = 1;
    public final static int VARIABLE_TYPE_GROUP = 1;
    public final static int VARIABLE_ARRAY_SIGN_GROUP = 2;
    public final static int VARIABLE_NAME_GROUP = 3;
    public final static int VARIABLE_VALUE_GROUP = 4;
    public final static int ASSIGNMENT_VARIABLE_GROUP = 1;
    public final static int GENERIC_VARIABLE_NAME_GROUP = 1;
    public final static int GENERIC_ARRAY_LOCATION_GROUP = 2;
    public final static int ASSIGNMENT_VALUE_GROUP = 4;
    public final static int METHOD_RETURN_TYPE_GROUP = 1;
    public final static int METHOD_RETURN_ARRAY_SIGN_GROUP = 2;
    public final static int METHOD_NAME_GROUP = 3;
    public final static int METHOD_PARAMETERS_GROUP = 4;
    public final static int METHOD_CALLING_NAME_GROUP = 1;
    public final static int METHOD_CALLING_PARAMETERS_GROUP = 2;
    public final static int TYPE_LOCATION = 1;
    public final static int ARRAY_LOCATION = 2;
    public final static int NAME_LOCATION = 3;
    public final static int EQUAL_SIGN_LOCATION = 4;
    public final static int VALUE_LOCATION = 5;
    public final static int SIMPLE_SCOPE_CONDITION_GROUP = 2;
    public final static String COMMA = "\\s*\\,\\s*";
    public final static String ARRAY_SIGN = "\\[\\s*\\]";
    public final static String ASSIGNMENT_ARRAY_SIGN = "\\[\\s*(.*)\\s*\\]";
    public final static String TYPES_REGEX = "(int|boolean|char|double|"
        + "String)\\s*(" + ARRAY_SIGN + ")?";
    public final static String METHOD_RETURN_TYPE_REGEX = "(int|boolean|"
        + "char|double|String|void)\\s*(" + ARRAY_SIGN + ")?";
    public final static String SIMPLE_SCOPE_NAME = "(if|while)";
    public final static String SIMPLE_SCOPE_REGEX = "^\\s*"
        + SIMPLE_SCOPE_NAME + "\\s*\\(\\s*(.*?)\\s*\\)\\s*\\{\\s*$";
    public final static String COMMENT_PATTERN = "\\s*//.*\\s*";
    public final static String METHOD_NAME_REGEX = "([A-Za-z]\\w*)";
    public final static String VARIABLE_NAME_REGEX = "([A-Za-z]\\w*|_\\w+)";
    public final static String VARIABLE_REGEX = "^\\s*(?:" + TYPES_REGEX
        + ")\\s+" + VARIABLE_NAME_REGEX + "\\s*(?:\\=\\s*(.*)\\s*)?\\;\\s*$";
    public final static String ASSIGNMENT_REGEX = "^\\s*("
        + VARIABLE_NAME_REGEX + "(?:\\s*" + ASSIGNMENT_ARRAY_SIGN
        + ")?)\\s*\\=\\s*(.*)\\s*\\;\\s*$";
    public final static String ARRAY_ASSIGNMENT_REGEX = "^\\{\\s*(.*)\\s*\\}$";
    public final static String DECLARATION_METHOD_REGEX = "^\\s*"
        + METHOD_RETURN_TYPE_REGEX + " +" + METHOD_NAME_REGEX + "\\s*"
        + "\\(\\s*(.*)\\s*\\)\\s*\\{\\s*\\;?\\s*$";
    public final static String METHOD_CALLING_REGEX = "^\\s*"
        + METHOD_NAME_REGEX + "\\s*" + "\\(\\s*(.*)\\s*\\)\\s*\\;?\\s*$";
    public final static String OPERAND_REGEX = "\\/|\\*|\\-|\\+";
    public final static String ANYTHING_BUT_OPERAND_REGEX =
        "([^\\/|\\*|\\-|\\+]+)";
    public final static String[] KEYWORDS = new String[] { "void", "int",
        "double", "boolean", "char", "String" };
    public final static String BLANK_LINE_PATTERN = "\\s*";
    public final static String RETURN_REGEX =
        "^\\s*return\\s*(.*)\\s*\\;\\s*$";
    public final static String END_OF_SCOPE_REGEX = "^\\s*\\}\\s*$";

    /**
     * This enum containing all the valid syntax for a line in a s-java file.
     *
     * @author roeia1
     *
     */
    public enum Syntax {
        VARIABLE(VARIABLE_REGEX),
        DECLARATION_METHOD(DECLARATION_METHOD_REGEX), CALLING_METHOD(
            METHOD_CALLING_REGEX), COMMENT(COMMENT_PATTERN), SIMPLE_SCOPE(
            SIMPLE_SCOPE_REGEX), BLANK_LINE(BLANK_LINE_PATTERN), RETURN(
            RETURN_REGEX), END_OF_SCOPE(END_OF_SCOPE_REGEX), ASSIGNMENT(
            ASSIGNMENT_REGEX);

        private final String syntax;
        private Matcher lineMatcher;

        /**
         * A data constructor.
         *
         * @param syntax
         *            - the valid syntax of the given line option.
         */
        Syntax(String syntax) {
            this.syntax = syntax;
        }

        public void setLineMatcher(String line) {
            this.lineMatcher = Pattern.compile(syntax).matcher(line);
        }

        public Matcher getLineMatcher() {
            return lineMatcher;
        }

    }

    /**
     * This static method checking a given s-java line from the file and return
     * the valid syntax value from the enum.
     *
     * @param line
     *            - the line from the s-java file.
     * @return The matched value from the enum.
     * @throws IllegalSyntaxException
     *             if there is a syntax error in the line.
     */
    public static Syntax getSyntax(String line) throws IllegalSyntaxException {
        for (Syntax syntax : Syntax.values()) {
            syntax.setLineMatcher(line);
            if (syntax.getLineMatcher().matches()) {
                return syntax;
            }
        }
        throw new IllegalSyntaxException("Syntax error");
    }

    public static void checkIfStartOrEndWithComma(String value)
        throws IllegalSyntaxException {
```

```java
        if (value.matches(COMMA + ".*") || value.matches(".*?" + COMMA)) {
            throw new IllegalSyntaxException(
                "Illegal comma placemnt syntax error)");
        }
    }
}
```

11 oop/ex7/main/Sjavac.java

```java
package oop.ex7.main;

import java.io.IOException;

import oop.ex7.main.scopes.ConditionalScope;
import oop.ex7.main.scopes.Scope;

/**
 * This class is the main class of program. This is the compiler, it is taking
 * the lines from the parser and prints the matching message.
 *
 * @author roeia1
 *
 */
public class Sjavac {

    public static Scope MAIN_SCOPE;

    /**
     * This is the main. Prints the matching message - 0 - if the code is legal.
     * 1 - if the code is illegal. 2- in case of IO errors.
     *
     * @param args
     */
    public static void main(String[] args) {
        MAIN_SCOPE = new Scope(null);
        // the try and catch structure. try if no exception was thrown, means
        // no error and print 0.
        try {
            Parser.Parse(args[0]);
            CheckScope(MAIN_SCOPE);
            System.out.println("0");
            // if met an IllegalSyntaxException or CompilationException than
            // prints 1 and the matching error message as was given in the
            // throw exception.
        } catch (IllegalSyntaxException | CompilationException e) {
            System.out.println("1");
            System.err.println(e.getMessage());
            e.printStackTrace();
            // another catch for the IO exception.
        } catch (IOException e) {
            System.out.println("2");
        }
    }

    /**
     * A static method that in each iteration checks all the s-java line of the
     * current scope, and later checks all of it's child.
     *
     * @param scopeToCheck
     *            - a given scope.
     * @throws CompilationException
     *             if there is a compilation error.
     */
    protected static void CheckScope(Scope scopeToCheck)
        throws CompilationException {
        // if the scope to check is an if/while scope than check the condition
        if (scopeToCheck instanceof ConditionalScope) {
            ((ConditionalScope) scopeToCheck).CheckCondition();
        }
        // sending to is valid for each scope and later for all the child
        // scopes
        for (SJavaLine currLine : scopeToCheck.getLineList()) {
            currLine.isValid(scopeToCheck);
        }
        for (Scope currScope : scopeToCheck.getChildScopeList()) {
            CheckScope(currScope);
        }
    }
}
```

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

```java
package oop.ex7.main.parameters;


/**
 * This exception indicates that a name is a keyword.
 *
 * @author roeia1
 *
 */
public class IllegalNameException extends IllegalParameterException {

    public IllegalNameException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

13 oop/ex7/main/parameters/IllegalParameterExceptio

```java
package oop.ex7.main.parameters;

import oop.ex7.main.IllegalSyntaxException;


/**
 * This exception indicates that there is a parameter syntax exception.
 *
 * @author roeia1
 *
 */
public class IllegalParameterException extends IllegalSyntaxException {

    public IllegalParameterException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

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

```java
package oop.ex7.main.parameters;

import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.SJavaSyntax;
import oop.ex7.main.Sjavac;
import oop.ex7.main.parameters.variables.Variable;
import oop.ex7.main.scopes.MethodScope;
import oop.ex7.main.scopes.Scope;

/**
 * This class represents a s-java line method calling. Extending the Parameter
 * class.
 *
 * @author roeia1
 *
 */
public class MethodCalling extends Parameter {

    private int numOfParameters;

    /**
     * Constructing a new method calling.
     *
     * @param name
     *            - the name of the method.
     * @param parameters
     *            - the string of all the parameters.
     * @throws IllegalSyntaxException
     *             if there is a syntax error in the method calling.
     */
    public MethodCalling(String name, String parameters)
        throws IllegalSyntaxException {
        super(name);
        numOfParameters = 0;
        if (!parameters.equals("")) {
            SJavaSyntax.checkIfStartOrEndWithComma(parameters);
            String[] parameterStrings = parameters.split(SJavaSyntax.COMMA);
            for (String currParameter : parameterStrings) {
                parameterList.add(Parameter.createParameter(currParameter));
                numOfParameters++;
            }

        }
    }

    public void isValid(Scope currentScope)
        throws ParameterCompilationException, MethodCallingNotExistException {
        MethodScope calledMethodScope = null;
        // Searching for the method declaration
        for (Scope currMethodScope : Sjavac.MAIN_SCOPE.getChildScopeList()) {
            if (((MethodScope) currMethodScope).getName().equals(
                this.getName())) {
                calledMethodScope = (MethodScope) currMethodScope;
                break;
            }
        }
        if (calledMethodScope == null) {
            throw new MethodCallingNotExistException(
                "Calling a method that don't exist error");
        }
        // Checking the method declaration and calling have the same number of
        // parameters
        if (calledMethodScope.getNumOfParameters() != this
            .getNumOfParameters()) {
            throw new MethodCallingException(
                "Mismatch number of variables in method calling error");
        } else {
            // Checking if each parameter in the calling match the type
            // in the declaration
            for (int currParameterIndex = 0; currParameterIndex <
                calledMethodScope.getNumOfParameters(); currParameterIndex++) {
                ((Variable) calledMethodScope.getLineList().get(
                    currParameterIndex)).checkAssignment(this
                    .getParameterList().get(currParameterIndex), currentScope);
            }
        }
    }

    public Variable find(Scope currentScope, boolean creationCheck)
        throws MethodCallingNotExistException {
        for (Scope currMethodScope : Sjavac.MAIN_SCOPE.getChildScopeList()) {
            if (((MethodScope) currMethodScope).getName().equals(
                this.getName())) {
                return ((MethodScope) currMethodScope).getReturnValue();
            }
        }
        throw new MethodCallingNotExistException("Method isn't found");
    }

    public int getNumOfParameters() {
        return numOfParameters;
    }
}
```

15 oop/ex7/main/parameters/MethodCallingExceptio

```java
package oop.ex7.main.parameters;

/**
 * This exception indicates that there is a compilation error in a method
 * calling.
 *
 * @author roeia1
 *
 */
public class MethodCallingException extends ParameterCompilationException {

    public MethodCallingException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

16 oop/ex7/main/parameters/MethodCallingNotExistE

```java
package oop.ex7.main.parameters;

/**
 * This exception indicates that the method calling isn't exist, a compilation
 * error.
 *
 * @author roeia1
 *
 */
public class MethodCallingNotExistException extends ParameterNotExistException {

    public MethodCallingNotExistException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

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

```java
package oop.ex7.main.parameters;

/**
 * This interface indicates that a parameter is operandable.
 *
 * @author roeia1
 *
 */
public interface Operandable {

}
```

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

```java
package oop.ex7.main.parameters;

import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import oop.ex7.main.CompilationException;
import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.SJavaLine;
import oop.ex7.main.SJavaSyntax;
import oop.ex7.main.parameters.variables.BooleanType;
import oop.ex7.main.parameters.variables.CharType;
import oop.ex7.main.parameters.variables.DoubleType;
import oop.ex7.main.parameters.variables.GenericType;
import oop.ex7.main.parameters.variables.IntegerType;
import oop.ex7.main.parameters.variables.StringType;
import oop.ex7.main.parameters.variables.Variable;
import oop.ex7.main.scopes.Scope;

/**
 * This abstract class is the parameter class, extends SJavaLine and designed to
 * work with initializing variables and method calling.
 *
 * @author roeia1
 *
 */
public abstract class Parameter extends SJavaLine {

    /** The regular expression of two parameters */
    protected final static String TWO_PARAMETERS_ASSIGNMENT_REGEX =
        "^\\s*\\-?\\s*(.+?)\\s*(?:\\/|\\*|\\-|\\+)\\s*(.*)\\s*$";
    private final static int FIRST_PARAMETER_GROUP = 1;
    private final static int SECOND_PARAMETER_GROUP = 2;
    private final static int ARRAY_ASSIGNMENT_PARAMETERS_GROUP = 1;

    protected ArrayList<Parameter> parameterList;
    private String name;

    /**
     * Constructing a raw parameter.
     */
    protected Parameter() {
        parameterList = new ArrayList<Parameter>();
    }

    /**
     * Constructing a new parameter.
     *
     * @param name
     *            - the name of the parameter.
     * @throws IllegalNameException
     *             if the parameter name isn't valid (a keyword).
     */
    protected Parameter(String name) throws IllegalNameException {
        this();
        checkIfNameIsKeyword(name);
        this.name = name;
    }

    /**
     * This enum include all the parameter types.
     *
     * @author roeia1
     *
     */
    private enum ParameterType {
        INTEGER(IntegerType.INTEGER_REGEX), DOUBLE(DoubleType.DOUBLE_REGEX),
        STRING(StringType.STRING_REGEX), CHAR(CharType.CHAR_REGEX), BOOLEAN(
            BooleanType.BOOLEAN_REGEX), METHOD_CALLING(
            SJavaSyntax.METHOD_CALLING_REGEX), GENERIC(
            GenericType.GENERIC_REGEX);

        private final String typeName;
        private Matcher typeMatcher;

        ParameterType(String type) {
            this.typeName = type;
        }

        public void setTypeMatcher(String line) {
            this.typeMatcher = Pattern.compile(typeName).matcher(line);
        }

        public Matcher getTypeMatcher() {
            return typeMatcher;
        }
    }

    /**
     * Getting the type of the parameter string.
     *
     * @param parameter
     *            - the parameter string.
     * @return The parameter type enum value.
     * @throws IllegalParameterException
     *             if the parameter string isn't match for any type.
     */
    public static ParameterType getType(String parameter)
        throws IllegalParameterException {
        for (ParameterType parameterType : ParameterType.values()) {
            parameterType.setTypeMatcher(parameter);
            if (parameterType.getTypeMatcher().matches()) {
                return parameterType;
            }
        }
        throw new IllegalParameterException("Illegal parameter syntax error");
    }

    /**
     * Creating one parameter.
     *
     * @param parameter
     *            - a given string which is the parameter.
     * @return The created parameter.
     * @throws IllegalParameterException
     *             if the parameter string isn't match for any type.
     * @throws IllegalSyntaxException
     *             if the parameter is method calling and there is a syntax
     *             error.
     */
    public static Parameter createOneParameter(String parameter)
        throws IllegalParameterException, IllegalSyntaxException {
        ParameterType parameterType = getType(parameter);
        switch (parameterType) {
        case INTEGER:
            return new IntegerType();
        case BOOLEAN:
            return new BooleanType();
        case CHAR:
            return new CharType();
        case DOUBLE:
            return new DoubleType();
        case STRING:
            return new StringType();
        case GENERIC:
            // Check if the generic is not an array
            if (parameterType.getTypeMatcher().group(
                SJavaSyntax.GENERIC_ARRAY_LOCATION_GROUP) == null) {
                return new GenericType(parameterType.getTypeMatcher().group(
                    SJavaSyntax.GENERIC_VARIABLE_NAME_GROUP));
            } else {
                return new GenericType(parameterType.getTypeMatcher().group(
                    SJavaSyntax.GENERIC_VARIABLE_NAME_GROUP), parameterType
                    .getTypeMatcher().group(
                        SJavaSyntax.GENERIC_ARRAY_LOCATION_GROUP));
            }
        case METHOD_CALLING:
            return new MethodCalling(parameterType.getTypeMatcher().group(
                SJavaSyntax.METHOD_CALLING_NAME_GROUP), parameterType
                .getTypeMatcher().group(
                    SJavaSyntax.METHOD_CALLING_PARAMETERS_GROUP));
        default:
            return null;
        }
    }

    /**
     * Creating one parameter, or two parameters (first, operand, second). If
     * two, adding the second parameter to the first parameter parameterList.
     *
     * @param value
     *            - a given value string.
     * @return The created parameter.
     * @throws IllegalParameterException
     *             if the parameter string isn't match for any type.
     * @throws IllegalSyntaxException
     *             if the parameter is method calling and there is a syntax
     *             error.
     */
    public static Parameter createParameter(String value)
        throws IllegalSyntaxException, IllegalParameterException {
        Pattern twoParametersPattern =
            Pattern.compile(TWO_PARAMETERS_ASSIGNMENT_REGEX);
        Matcher twoParametersMatcher = twoParametersPattern.matcher(value);
        if (twoParametersMatcher.matches()) {
            Parameter newParameter =
                createOneParameter(twoParametersMatcher
                    .group(FIRST_PARAMETER_GROUP));
            newParameter.getParameterList().add(
                createOneParameter(twoParametersMatcher
                    .group(SECOND_PARAMETER_GROUP)));
            return newParameter;
        } else {
            return createOneParameter(value);
        }
    }

    /**
     * Creating an ArrayList of parameters.
     *
     * @param assignmentValue
     *            - the assignment value string of an array.
     * @return The ArrayList of parameters.
     * @throws IllegalSyntaxException
     *             if there is a syntax error in the parameters string.
```

```java
     * @throws IllegalParameterException
     *              if one of the parameters string isn't match for any type.
     */
    public static ArrayList<Parameter> arrayAssignment(String assignmentValue)
        throws IllegalSyntaxException, IllegalParameterException {
        Pattern arrayAssignmentPattern =
            Pattern.compile(SJavaSyntax.ARRAY_ASSIGNMENT_REGEX);
        Matcher arrayAssignmentMatcher =
            arrayAssignmentPattern.matcher(assignmentValue);
        // Checking if the array pattern matches
        if (arrayAssignmentMatcher.matches()) {
            ArrayList<Parameter> arrayAssignmentParameters =
                new ArrayList<Parameter>();
            // Checking if the array assignment not empty {}
            if (!arrayAssignmentMatcher.group(
                ARRAY_ASSIGNMENT_PARAMETERS_GROUP).equals("")) {
                SJavaSyntax.checkIfStartOrEndWithComma(arrayAssignmentMatcher
                    .group(ARRAY_ASSIGNMENT_PARAMETERS_GROUP));
                // Splitting the parameters
                String[] parameterStrings =
                    arrayAssignmentMatcher.group(
                        ARRAY_ASSIGNMENT_PARAMETERS_GROUP).split(
                        SJavaSyntax.COMMA);
                // Creating each parameter
                for (String currArrayParameter : parameterStrings) {
                    arrayAssignmentParameters
                        .add(createParameter(currArrayParameter));
                }

            }
            return arrayAssignmentParameters;
        } else {
            // If array pattern don't match
            throw new IllegalSyntaxException("Illegal assignment syntax error");
        }
    }

    /**
     * Checking if a string is a restricted keyword.
     *
     * @param value
     *             - A given String.
     * @throws IllegalNameException
     *              being thrown if the given String is equal to one of the
     *              keywords above.
     */
    public static void checkIfNameIsKeyword(String value)
        throws IllegalNameException {
        for (String Keyword : SJavaSyntax.KEYWORDS) {
            if (value.equals(Keyword)) {
                throw new IllegalNameException(
                    "Parameter name is a keyword error");
            }
        }
    }

    /**
     * Finding a parameter.
     *
     * @param currentScope
     *             - the current scope.
     * @return The found parameter. If it's a method calling then returning a
     *         variable represents the return type of the method
     * @throws NotExistVariableException
     *              if the variable isn't exist.
     * @throws NotMethodCallingException
     *              if the method isn't exist.
     * @throws CompilationException
     */
    public abstract Variable find(Scope currentScope, boolean creationCheck)
        throws ParameterNotExistException;

    public ArrayList<Parameter> getParameterList() {
        return parameterList;
    }

    public String getName() {
        return name;
    }

}
```

19 oop/ex7/main/parameters/ParameterCompilationEx

```
package oop.ex7.main.parameters;

import oop.ex7.main.CompilationException;


/**
 * This exception indicates that there is a compilation error in a parameter.
 *
 * @author roeia1
 *
 */
public class ParameterCompilationException extends CompilationException {

    public ParameterCompilationException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

20 oop/ex7/main/parameters/ParameterNotExistExce

```java
package oop.ex7.main.parameters;

/**
 * This exception indicates that a parameter isn't exist, a compilation error.
 *
 * @author roeia1
 *
 */
public class ParameterNotExistException extends ParameterCompilationException {

    public ParameterNotExistException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

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

```java
package oop.ex7.main.scopes;

import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.parameters.IllegalParameterException;
import oop.ex7.main.parameters.MethodCalling;
import oop.ex7.main.parameters.Parameter;
import oop.ex7.main.parameters.ParameterNotExistException;
import oop.ex7.main.parameters.variables.BooleanType;
import oop.ex7.main.parameters.variables.GenericType;
import oop.ex7.main.parameters.variables.Variable;

/**
 * This class represents a scope containing a condition. (if / while scope).
 * Extending the Scope class.
 *
 * @author roeia1
 *
 */
public class ConditionalScope extends Scope {

    private Parameter condition;

    /**
     * A data constructor.
     *
     * @param fatherScope
     *            - the father scope of this scope.
     * @param condition
     *            - the condition of this scope.
     * @throws IllegalParameterException
     *             if the condition isn't match for any parameter type.
     * @throws IllegalSyntaxException
     *             if the condition is a method calling and there is a syntax
     *             error.
     */
    public ConditionalScope(Scope fatherScope, String condition)
        throws IllegalParameterException, IllegalSyntaxException {
        super(fatherScope);
        this.condition = Parameter.createOneParameter(condition);
    }

    /**
     * Checking if the condition is a boolean parameter.
     *
     * @throws MismatchConditionTypeException
     *             if the condition isn't a boolean parameter.
     * @throws ParameterNotExistException
     *             if the condition is a parameter and it doesn't exist.
     */
    public void CheckCondition() throws MismatchConditionTypeException,
        ParameterNotExistException {
        if (!(this.condition instanceof BooleanType)) {
            if (this.condition instanceof GenericType
                || this.condition instanceof MethodCalling) {
                Variable foundVariable =
                    this.condition.find(this.getFatherScope(), false);
                if (!(foundVariable instanceof BooleanType)) {
                    throw new MismatchConditionTypeException(
                        "Mismatch condition type error");
                }
                if (this.condition instanceof GenericType
                    && !foundVariable.isInitialized()) {
                    throw new MismatchConditionTypeException(
                        "Boolean variable not initialized in the condition");
                }
            } else {
                throw new MismatchConditionTypeException(
                    "Mismatch condition type error");
            }
        }
    }

    public Parameter getCondition() {
        return condition;
    }
}
```

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

```java
package oop.ex7.main.scopes;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.SJavaLine;
import oop.ex7.main.SJavaSyntax;
import oop.ex7.main.parameters.IllegalNameException;
import oop.ex7.main.parameters.Parameter;
import oop.ex7.main.parameters.variables.Variable;
import oop.ex7.main.parameters.variables.VariableFactory;


/**
 * This class represents a method scope in the s-java file. Extends the Scope
 * class.
 *
 * @author roeia1
 *
 */
public class MethodScope extends Scope {

    private final static String DECLARATION_PARAMETER_REGEX = "(?:\\s*"
        + SJavaSyntax.TYPES_REGEX + "\\s*" + SJavaSyntax.METHOD_NAME_REGEX
        + "\\s*)";
    private final static int PARAMETER_TYPE_GROUP = 1;
    private final static int PARAMETER_ARRAY_SIGN_GROUP = 2;
    private final static int PARAMETER_NAME_GROUP = 3;
    private String name;
    private Variable returnType;
    private int numOfParameters;

    /**
     * A data constructor.
     *
     * @param fatherScope
     *            - the father scope of this method scope.
     * @param returnType
     *            - the return type of the method.
     * @param arraySign
     *            - the return type array sign.
     * @param name
     *            - the method name.
     * @param parameters
     *            - the declaration parameters of the method.
     * @throws IllegalSyntaxException
     *             if there is a syntax error in the method scope.
     */
    public MethodScope(Scope fatherScope, String returnType, String arraySign,
        String name, String parameters) throws IllegalSyntaxException {
        super(fatherScope);
        Parameter.checkIfNameIsKeyword(name);
        this.name = name;
        if (returnType.equals("void")) {
            this.returnType = null;
        } else {
            this.returnType =
                VariableFactory.createVariable(returnType, arraySign, null,
                        null);
        }
        /*
         * Dividing the given parameters string, and creating each parameter
         * while counting them.
         */
        this.numOfParameters = 0;
        if (!parameters.equals("")) {
            SJavaSyntax.checkIfStartOrEndWithComma(parameters);
            String[] parametersArray = parameters.split(SJavaSyntax.COMMA);
            Pattern parameterPattern =
                Pattern.compile(DECLARATION_PARAMETER_REGEX);
            Matcher parameterMatcher;
            for (String currParameter : parametersArray) {
                parameterMatcher = parameterPattern.matcher(currParameter);
                if (parameterMatcher.matches()) {
                    Variable declarationParameter =
                        VariableFactory
                            .createVariable(parameterMatcher
                                .group(PARAMETER_TYPE_GROUP), parameterMatcher
                                .group(PARAMETER_ARRAY_SIGN_GROUP),
                                parameterMatcher.group(PARAMETER_NAME_GROUP),
                                null);
                    // Check if the variable name already exist
                    // in the declaration
                    for (SJavaLine currLine : this.getLineList()) {
                        if (declarationParameter.getName().equals(
                            ((Variable) currLine).getName())) {
                            throw new IllegalNameException(""
                                + "Already exist variable error");
                        }
                    }
                    // Making the declaration parameter initialized by
                    // adding to it a raw parameter from the same type
                    declarationParameter.getParameterList().add(
                        VariableFactory.createVariable(parameterMatcher
                            .group(PARAMETER_TYPE_GROUP), null, null, null));
                    this.getLineList().add(declarationParameter);
                    this.numOfParameters++;
                } else {
                    throw new IllegalSyntaxException(
                        "Illegal parameter of method decleration error");
                }
            }

        }
    }

    public int getNumOfParameters() {
        return numOfParameters;
    }

    public String getName() {
        return name;
    }

    public Variable getReturnValue() {
        return returnType;
    }

}
```

23 oop/ex7/main/scopes/MismatchConditionTypeExce

```java
package oop.ex7.main.scopes;

import oop.ex7.main.CompilationException;


/**
 * This exception indicates that the condition is not a boolean type.
 *
 * @author roeia1
 *
 */
public class MismatchConditionTypeException extends CompilationException {

    public MismatchConditionTypeException(String errorMessage) {
        super(errorMessage);
    }

    private static final long serialVersionUID = 1L;

}
```

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

```java
package oop.ex7.main.scopes;

import java.util.ArrayList;

import oop.ex7.main.SJavaLine;

/**
 * This class represents a scope in the s-java file.
 *
 * @author roeia1
 *
 */
public class Scope {

    private ArrayList<SJavaLine> lineList;
    private ArrayList<Scope> childScopeList;
    private Scope fatherScope;

    /**
     * A data constructor.
     *
     * @param fatherScope
     *            - The father scope of this scope.
     */
    public Scope(Scope fatherScope) {
        this.fatherScope = fatherScope;
        this.lineList = new ArrayList<SJavaLine>();
        this.childScopeList = new ArrayList<Scope>();
    }

    public Scope getFatherScope() {
        return fatherScope;
    }

    public ArrayList<SJavaLine> getLineList() {
        return lineList;
    }

    public ArrayList<Scope> getChildScopeList() {
        return childScopeList;
    }
}
```

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

```java
package oop.ex7.main.parameters.variables;

import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.parameters.IllegalNameException;

/**
 * This class is the BooleanType extends the variable class (true\false).
 *
 * @author roeia1
 *
 */
public class BooleanType extends Variable {

    /** The regular expression of a raw boolean */
    public final static String BOOLEAN_REGEX = "^\\s*true|false\\s*$";

    /**
     * Constructing a raw boolean.
     */
    public BooleanType() {
        super();
    }

    /**
     * Constructing a return value boolean for a method.
     *
     * @param isArray
     *            - represents if the return value boolean is an array.
     */
    public BooleanType(boolean isArray) {
        super(isArray);
    }

    /**
     * Constructing a new boolean variable.
     *
     * @param name
     *            - the name of the boolean variable.
     * @param isArray
     *            - represents if the new boolean variable is an array.
     * @throws IllegalNameException
     *             if the boolean variable name isn't valid (a keyword).
     */
    public BooleanType(String name, boolean isArray)
        throws IllegalNameException {
        super(name, isArray);
    }

    /**
     * Constructing a new boolean variable with an assignment value.
     *
     * @param name
     *            - the name of the boolean variable.
     * @param isArray
     *            - represents if the new boolean variable is an array.
     * @param assignmentValue
     *            - the assignment value string.
     * @throws IllegalNameException
     *             if the boolean variable name isn't valid (a keyword).
     * @throws IllegalSyntaxException
     *             if there is a syntax error with the assignment string.
     */
    public BooleanType(String name, boolean isArray, String assignmentValue)
        throws IllegalNameException, IllegalSyntaxException {
        super(name, isArray, assignmentValue);
    }
}
```

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

```java
package oop.ex7.main.parameters.variables;

import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.parameters.IllegalNameException;

/**
 * This class is the CharType extends the variable class.
 *
 * @author roeia1
 *
 */
public class CharType extends Variable {

    /** The regular expression of a raw char */
    public final static String CHAR_REGEX = "^\\s*\\'.?\\'\\s*$";

    /**
     * Constructing a raw char.
     */
    public CharType() {
        super();
    }

    /**
     * Constructing a return value char for a method.
     *
     * @param isArray
     *            - represents if the return value char is an array.
     */
    public CharType(boolean isArray) {
        super(isArray);
    }

    /**
     * Constructing a new char variable.
     *
     * @param name
     *            - the name of the char variable.
     * @param isArray
     *            - represents if the new char variable is an array.
     * @throws IllegalNameException
     *             if the char variable name isn't valid (a keyword).
     */
    public CharType(String name, boolean isArray) throws IllegalNameException {
        super(name, isArray);
    }

    /**
     * Constructing a new char variable with an assignment value.
     *
     * @param name
     *            - the name of the char variable.
     * @param isArray
     *            - represents if the new char variable is an array.
     * @param assignmentValue
     *            - the assignment value string.
     * @throws IllegalNameException
     *             if the char variable name isn't valid (a keyword).
     * @throws IllegalSyntaxException
     *             if there is a syntax error with the assignment string.
     */
    public CharType(String name, boolean isArray, String assignmentValue)
        throws IllegalNameException, IllegalSyntaxException {
        super(name, isArray, assignmentValue);
    }
}
```

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

```java
package oop.ex7.main.parameters.variables;

import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.parameters.IllegalNameException;
import oop.ex7.main.parameters.Operandable;

/**
 * This class is the DoubleType extends the variable class (5,5.2), and it is
 * also implements Operandable interface (5+3.2).
 *
 * @author roeia1
 *
 */
public class DoubleType extends Variable implements Operandable {

    /** The regular expression of a raw double */
    public final static String DOUBLE_REGEX = "^\\s*\\-?\\s*\\d+\\.\\d+\\s*$";

    /**
     * Constructing a raw double.
     */
    public DoubleType() {
        super();
    }

    /**
     * Constructing a return value double for a method.
     *
     * @param isArray
     *            - represents if the return value double is an array.
     */
    public DoubleType(boolean isArray) {
        super(isArray);
    }

    /**
     * Constructing a new double variable.
     *
     * @param name
     *            - the name of the double variable.
     * @param isArray
     *            - represents if the new double variable is an array.
     * @throws IllegalNameException
     *             if the double variable name isn't valid (a keyword).
     */
    public DoubleType(String name, boolean isArray)
        throws IllegalNameException {
        super(name, isArray);
    }

    /**
     * Constructing a new double variable with an assignment value.
     *
     * @param name
     *            - the name of the double variable.
     * @param isArray
     *            - represents if the new double variable is an array.
     * @param assignmentValue
     *            - the assignment value string.
     * @throws IllegalNameException
     *             if the double variable name isn't valid (a keyword).
     * @throws IllegalSyntaxException
     *             if there is a syntax error with the assignment string.
     */
    public DoubleType(String name, boolean isArray, String assignmentValue)
        throws IllegalNameException, IllegalSyntaxException {
        super(name, isArray, assignmentValue);
    }
}
```

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

```java
package oop.ex7.main.parameters.variables;

import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.SJavaSyntax;
import oop.ex7.main.parameters.IllegalNameException;
import oop.ex7.main.parameters.IllegalParameterException;
import oop.ex7.main.parameters.Parameter;

/**
 * This class is the GenericType variable, extends the variable class.
 * GenericType is a variable which his type is unknown in the syntax check
 * phase.
 *
 * @author roeia1
 *
 */
public class GenericType extends Variable {

    /** The regular expression of a generic variable */
    public final static String GENERIC_REGEX = "^\\s*\\-?\\s*"
        + SJavaSyntax.VARIABLE_NAME_REGEX + "\\s*(?:"
        + SJavaSyntax.ASSIGNMENT_ARRAY_SIGN + ")?\\s*$";
    private Parameter arrayLocationParameter;

    /**
     * Constructing a new generic variable.
     *
     * @param name
     *            - the name of the generic variable.
     * @throws IllegalNameException
     *             if the generic variable name isn't valid (a keyword).
     */
    public GenericType(String name) throws IllegalNameException {
        super(name, false);
    }

    /**
     * Constructing a new generic variable that's inside an array (abc[1]).
     *
     * @param name
     *            - the name of the generic variable.
     * @param arrayLocationParameter
     *            - the location inside the array.
     * @throws IllegalSyntaxException
     *             if the array location parameter is a method calling with a
     *             syntax error.
     * @throws IllegalParameterException
     *             if the array location parameter string isn't match for any
     *             type.
     * @throws IllegalArrayLocationException
     *             if the array location is a raw integer that is negative.
     */
    public GenericType(String name, String arrayLocationParameter)
        throws IllegalParameterException, IllegalSyntaxException,
        IllegalArrayLocationException {
        super(name, true);
        this.arrayLocationParameter =
            Parameter.createParameter(arrayLocationParameter);
        // Checking if the location is a negative number
        if (this.arrayLocationParameter instanceof IntegerType
            && this.arrayLocationParameter.getParameterList().isEmpty()
            && arrayLocationParameter.contains("-")) {
            throw new IllegalArrayLocationException(
                "Illegal array location error");
        }

    }

    public Parameter getArrayLocationParameter() {
        return arrayLocationParameter;
    }
}
```

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

```java
package oop.ex7.main.parameters.variables;

/**
 * This exception indicates there is an error in the array location (abc[-1]).
 *
 * @author roeia1
 *
 */
public class IllegalArrayLocationException extends
    IllegalVariableSyntaxException {

    public IllegalArrayLocationException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

30 oop/ex7/main/parameters/variables/IllegalVariable

```java
package oop.ex7.main.parameters.variables;

import oop.ex7.main.parameters.IllegalParameterException;

/**
 * This exception indicates that there is a syntax error in a variable.
 *
 * @author roeia1
 *
 */
public class IllegalVariableSyntaxException extends IllegalParameterException {

    public IllegalVariableSyntaxException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

31 oop/ex7/main/parameters/variables/IllegalVariable

```java
package oop.ex7.main.parameters.variables;

import oop.ex7.main.IllegalSyntaxException;


/**
 * This exception indicates that there is a syntax error in the variable type.
 *
 * @author roeia1
 *
 */
public class IllegalVariableTypeException extends IllegalSyntaxException {

    /**
     * super the constructor with error message.
     */
    public IllegalVariableTypeException(String errorMessage) {
        super(errorMessage);
    }

    private static final long serialVersionUID = 1L;

}
```

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

```java
package oop.ex7.main.parameters.variables;

import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.parameters.IllegalNameException;

/**
 * This class represents the integer type variable, extends from double type.
 *
 * @author roeia1
 *
 */
public class IntegerType extends DoubleType {

    /** The regular expression of a raw integer */
    public static final String INTEGER_REGEX = "^\\s*\\-?\\s*\\d+\\s*$";

    /**
     * Constructing a raw integer.
     */
    public IntegerType() {
        super();
    }

    /**
     * Constructing a return value integer for a method.
     *
     * @param isArray
     *            - represents if the return value integer is an array.
     */
    public IntegerType(boolean isArray) {
        super(isArray);
    }

    /**
     * Constructing a new integer variable.
     *
     * @param name
     *            - the name of the integer variable.
     * @param isArray
     *            - represents if the new integer variable is an array.
     * @throws IllegalNameException
     *             if the integer variable name isn't valid (a keyword).
     */
    public IntegerType(String name, boolean isArray)
        throws IllegalNameException {
        super(name, isArray);
    }

    /**
     * Constructing a new integer variable with an assignment value.
     *
     * @param name
     *            - the name of the integer variable.
     * @param isArray
     *            - represents if the new integer variable is an array.
     * @param assignmentValue
     *            - the assignment value string.
     * @throws IllegalNameException
     *             if the integer variable name isn't valid (a keyword).
     * @throws IllegalSyntaxException
     *             if there is a syntax error with the assignment string.
     */
    public IntegerType(String name, boolean isArray, String assignmentValue)
        throws IllegalNameException, IllegalSyntaxException {
        super(name, isArray, assignmentValue);
    }

}
```

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

```java
package oop.ex7.main.parameters.variables;

/**
 * This exception indicates that there is a method calling return variable
 * mismatch to the assigned variable.
 *
 * @author roeia1
 *
 */
public class MethodCallingTypeMismatchException extends
    VariableCompilationException {

    public MethodCallingTypeMismatchException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

34 oop/ex7/main/parameters/variables/NotOperandab

```java
package oop.ex7.main.parameters.variables;

/**
 * This exception indicates that a method calling return variable isn't
 * operandable while using it with operand.
 *
 * @author roeia1
 *
 */
public class NotOperandableMethodCallingException extends
    VariableCompilationException {

    public NotOperandableMethodCallingException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

35 oop/ex7/main/parameters/variables/NotOperandab

```java
package oop.ex7.main.parameters.variables;

/**
 * This exception indicates that a variable isn't operandable while using it
 * with operand.
 *
 * @author roeia1
 *
 */
public class NotOperandableVariableException extends
    VariableCompilationException {

    public NotOperandableVariableException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

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

```java
package oop.ex7.main.parameters.variables;

import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.parameters.IllegalNameException;

/**
 * This class represents a string variable, extends from variable class.
 *
 * @author roeia1
 *
 */
public class StringType extends Variable {

    /** The regular expression of a raw String */
    public final static String STRING_REGEX = "^\\s*\".*\"\\s*$";

    /**
     * Constructing a raw String.
     */
    public StringType() {
        super();
    }

    /**
     * Constructing a return value String for a method.
     *
     * @param isArray
     *            - represents if the return value String is an array.
     */
    public StringType(boolean isArray) {
        super(isArray);
    }

    /**
     * Constructing a new String variable.
     *
     * @param name
     *            - the name of the String variable.
     * @param isArray
     *            - represents if the new String variable is an array.
     * @throws IllegalNameException
     *             if the String variable name isn't valid (a keyword).
     */
    public StringType(String name, boolean isArray)
        throws IllegalNameException {
        super(name, isArray);
    }

    /**
     * Constructing a new String variable with an assignment value.
     *
     * @param name
     *            - the name of the String variable.
     * @param isArray
     *            - represents if the new String variable is an array.
     * @param assignmentValue
     *            - the assignment value string.
     * @throws IllegalNameException
     *             if the String variable name isn't valid (a keyword).
     * @throws IllegalSyntaxException
     *             if there is a syntax error with the assignment string.
     */
    public StringType(String name, boolean isArray, String assignmentValue)
        throws IllegalNameException, IllegalSyntaxException {
        super(name, isArray, assignmentValue);
    }
}
```

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

```java
package oop.ex7.main.parameters.variables;

/**
 * This exception indicates that an uninitialized variable used for assignment,
 * compilation error.
 *
 * @author Roei
 *
 */
public class UninitializedVariableException extends
    VariableCompilationException {

    public UninitializedVariableException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

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

```java
package oop.ex7.main.parameters.variables;

import java.util.ArrayList;

import oop.ex7.main.AssignmentLine;
import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.ReturnLine;
import oop.ex7.main.SJavaLine;
import oop.ex7.main.Sjavac;
import oop.ex7.main.parameters.IllegalNameException;
import oop.ex7.main.parameters.MethodCalling;
import oop.ex7.main.parameters.Operandable;
import oop.ex7.main.parameters.Parameter;
import oop.ex7.main.parameters.ParameterCompilationException;
import oop.ex7.main.parameters.ParameterNotExistException;
import oop.ex7.main.scopes.Scope;

/**
 * This abstract class represent all of the legitimate variables in s-java. It
 * extends the parameter class.
 *
 * @author roeia1
 *
 */
public abstract class Variable extends Parameter {

    private boolean isArray;

    /**
     * Constructing a raw variable.
     */
    protected Variable() {
        super();
    }

    /**
     * Constructing a new variable.
     *
     * @param name
     *             - the name of the variable.
     * @throws IllegalNameException
     *              if the variable name isn't valid (a keyword).
     */
    protected Variable(String name) throws IllegalNameException {
        super(name);
    }

    /**
     * Constructing a new variable.
     *
     * @param isArray
     *             - represents if the variable is an array.
     */
    protected Variable(boolean isArray) {
        this.isArray = isArray;
    }

    /**
     * Constructing a new variable.
     *
     * @param name
     *            - the name of the variable.
     * @param isArray
     *            - represents if the variable is an array.
     * @throws IllegalNameException
     *             if the variable name isn't valid (a keyword).
     */
    protected Variable(String name, boolean isArray)
        throws IllegalNameException {
        super(name);
        this.isArray = isArray;
    }

    /**
     * Constructing a new variable with an assignment value.
     *
     * @param name
     *            - the name of the variable.
     * @param isArray
     *            - represents if the variable is an array.
     * @param assignmentValue
     *            - the assignment value string.
     * @throws IllegalSyntaxException
     *             if there is a syntax error with the assignment string.
     */
    protected Variable(String name, boolean isArray, String assignmentValue)
        throws IllegalSyntaxException {
        super(name);
        this.isArray = isArray;
        if (isArray) {
            this.parameterList = arrayAssignment(assignmentValue);
        } else {
            this.parameterList.add(createParameter(assignmentValue));
        }
    }

    public void isValid(Scope currentScope)
        throws VariableAlreadyExistException, ParameterCompilationException {
        try {
            this.find(currentScope, true);
            throw new VariableAlreadyExistException(
                "Already exist variable error");
        } catch (VariableNotExistException e) {
            for (Parameter currParameter : this.getParameterList()) {
                this.checkAssignment(currParameter, currentScope);
            }
        }
    }

    /**
     * Checking if the assignment of the parameter input is valid for this
     * variable.
     *
     * @param parameter
     *            - the assigning parameter.
     * @param currentScope
     *            - the current scope of the variable.
     * @throws ParameterNotExistException
     *             if the parameter doesn't exist.
     * @throws NotOperandableVariableException
     *             if assigning parameters with operand and one of them is a
     *             variable that isn't operandable.
     * @throws VariableTypeMismatchException
     *             if assigning a variable that isn't from the same type as the
     *             assigned one.
     * @throws UninitializedVariableException
     *             if assigning a variable that isn't initialized.
     * @throws VoidMethodException
     *              if assigning a method that returns void.
     * @throws NotOperandableMethodCallingException
     *              if assigning parameters with operand and one of them is a
     *              method calling that returns a variable that isn't
     *              operandable.
     * @throws MethodCallingTypeMismatchException
     *              if assigning a method calling that returns a variable that
     *              isn't from the same type as the assigned one.
     * @throws ParameterCompilationException
     *              if the parameter is a method calling and there is a
     *              compilation error when checking validation.
     */
    public void checkAssignment(Parameter parameter, Scope currentScope)
        throws ParameterNotExistException, NotOperandableVariableException,
        VariableTypeMismatchException, UninitializedVariableException,
        VoidMethodException, NotOperandableMethodCallingException,
        MethodCallingTypeMismatchException, ParameterCompilationException {
        ArrayList<Parameter> parameters = new ArrayList<Parameter>();
        Variable foundVariable;
        parameters.add(parameter);
        // Checking if the assignment is with two parameters with operand
        if (parameter instanceof MethodCalling) {
            if (parameter.getParameterList().size() >
            ((MethodCalling) parameter).getNumOfParameters()) {
                parameters.add(parameter.getParameterList().get(
                    ((MethodCalling) parameter).getNumOfParameters()));
            }
        } else if (!parameter.getParameterList().isEmpty()) {
            parameters.add(parameter.getParameterList().get(0));
        }
        for (Parameter currParameter : parameters) {
            if (currParameter instanceof GenericType) {
                foundVariable = currParameter.find(currentScope, false);
                if (parameters.size() == 2
                    && !(foundVariable instanceof Operandable)) {
                    throw new NotOperandableVariableException(
                        "Assignment using operand with a parameter that"
                        + " is not operanable");
                }
                if (!this.getClass().isInstance(foundVariable)) {
                    throw new VariableTypeMismatchException(
                        "Assignment of a variable not from same type");
                }
                if (!foundVariable.isInitialized()) {
                    throw new UninitializedVariableException(
                        "Assignment of an uninitialized variable");
                }
            } else if (currParameter instanceof MethodCalling) {
                ((MethodCalling) currParameter).isValid(currentScope);
                foundVariable = currParameter.find(currentScope, false);
                if (foundVariable == null) {
                    throw new VoidMethodException(
                        "Assignment using a method calling that return void");
                }
                if (parameters.size() == 2
                    && !(foundVariable instanceof Operandable)) {
                    throw new NotOperandableMethodCallingException(
                        "Assignment using operand with a parameter that is"
                        + " not operanable");
                }
                if (!this.getClass().isInstance(foundVariable)) {
                    throw new MethodCallingTypeMismatchException(
                        "Assignment of a method calling that not returning "
                        + "same type");
                }
            } else if (parameters.size() == 2
                && !(this instanceof Operandable)) {
```

```java
                throw new NotOperandableVariableException(
                    "Assignment using operand with a parameter that is not "
                    + "operanable");
            } else if (!this.getClass().isInstance(currParameter)) {
                throw new VariableTypeMismatchException(
                    "Assignment of a raw variable that not from the same "
                    + "type");
            }
        }
    }

    public Variable find(Scope currentScope, boolean creationCheck)
        throws VariableNotExistException {
        for (SJavaLine currLine : currentScope.getLineList()) {
            // check for all the cases which there are not the return line
            if (!(currLine instanceof ReturnLine)) {
                // check if the variable exists
                if (this == currLine
                    || (currLine instanceof Variable && ((Variable) currLine)
                        .getParameterList().contains(this))
                    || (currLine instanceof AssignmentLine &&
                        ((AssignmentLine) currLine)
                        .getAssignedGeneric() == this)) {
                    break;
                }
                if (currLine instanceof Variable
                    && ((Variable) currLine).getName().equals(this.getName())) {
                    return (Variable) currLine;
                }
            }
        }
        // going to the father scope.
        currentScope = currentScope.getFatherScope();
        if (creationCheck) {
            if (currentScope != null && currentScope != Sjavac.MAIN_SCOPE) {
                return (Variable) recursiveFind(currentScope, creationCheck);
            }
        } else {
            if (currentScope != null) {
                return (Variable) recursiveFind(currentScope, creationCheck);
            }
        }
        throw new VariableNotExistException("Variable doesn't exist");
    }

    private Parameter recursiveFind(Scope currentScope, boolean creationCheck)
        throws VariableNotExistException {
        // running on the lines of current scope.
        for (SJavaLine currLine : currentScope.getLineList()) {
            // if found it than return the current line as variable or
            // parameter depends on the instance it has.
            if (currLine instanceof Variable
                && ((Variable) currLine).getName().equals(this.getName())) {
                return (Variable) currLine;
            }
        }
        // going to the father scope.
        currentScope = currentScope.getFatherScope();
        if (creationCheck) {
            if (currentScope != Sjavac.MAIN_SCOPE) {
                return (Variable) recursiveFind(currentScope, creationCheck);
            }
        } else {
            if (currentScope != null) {
                return (Variable) recursiveFind(currentScope, creationCheck);
            }
        }
        throw new VariableNotExistException("Variable doesn't exist");
    }

    /**
     * Checking if a variable is initialized.
     *
     * @return True if the variable initialized, false otherwise.
     */
    public boolean isInitialized() {
        return !this.getParameterList().isEmpty();
    }

    public boolean isArray() {
        return isArray;
    }
}
```

39 oop/ex7/main/parameters/variables/VariableAlread

```java
package oop.ex7.main.parameters.variables;

/**
 * This exception extends the CompilationException and being thrown when a
 * variable is already exist.
 *
 * @author roeia1
 *
 */
public class VariableAlreadyExistException extends
    VariableCompilationException {

    /**
     * super the constructor with error message.
     *
     * @param errorMessage
     */
    public VariableAlreadyExistException(String errorMessage) {
        super(errorMessage);
    }

    private static final long serialVersionUID = 1L;

}
```

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

```java
package oop.ex7.main.parameters.variables;

import oop.ex7.main.parameters.ParameterCompilationException;

/**
 * This exception indicates that there is a compilation error with the variable.
 *
 * @author roeia1
 *
 */
public class VariableCompilationException extends
    ParameterCompilationException {

    public VariableCompilationException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

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

```java
package oop.ex7.main.parameters.variables;

import oop.ex7.main.IllegalSyntaxException;
import oop.ex7.main.parameters.IllegalNameException;

/**
 * This class represents the variable factory.
 *
 * @author roeia1
 *
 */
public class VariableFactory {

    /**
     * Creating a variable.
     *
     * @param type
     *            - the variable type.
     * @param arraySign
     *            - the array sign.
     * @param name
     *            - the name of the variable.
     * @param assignmentvalue
     *            - the assignment value of the new variable.
     * @return The new variable.
     * @throws IllegalNameException
     *             if the name of the variable is a keyword.
     * @throws IllegalSyntaxException
     *             if the assignment string has a syntax error.
     */
    public static Variable createVariable(String type, String arraySign,
        String name, String assignmentvalue) throws IllegalNameException,
        IllegalSyntaxException {
        boolean isArray = false;
        // Checking if the variable is array
        if (arraySign != null) {
            isArray = !(arraySign.equals(""));
        }
        Variable newVariable;
        // Creating a variable parameter
        switch (type) {
        case "int":
            if (name == null) {
                if (arraySign == null) {
                    // Creating a raw parameter
                    newVariable = new IntegerType();
                } else {
                    // Creating a return type variable for a method declaration
                    newVariable = new IntegerType(isArray);
                }
            } else if (assignmentvalue != null) {
                newVariable = new IntegerType(name, isArray, assignmentvalue);
            } else {
                newVariable = new IntegerType(name, isArray);
            }
            break;
        case "boolean":
            if (name == null) {
                if (arraySign == null) {
                    // Creating a raw parameter
                    newVariable = new BooleanType();
                } else {
                    // Creating a return type variable for a method declaration
                    newVariable = new BooleanType(isArray);
                }
            } else if (assignmentvalue != null) {
                newVariable = new BooleanType(name, isArray, assignmentvalue);
            } else {
                newVariable = new BooleanType(name, isArray);
            }
            break;
        case "String":
            if (name == null) {
                if (arraySign == null) {
                    // Creating a raw parameter
                    newVariable = new StringType();
                } else {
                    // Creating a return type variable for a method declaration
                    newVariable = new StringType(isArray);
                }
            } else if (assignmentvalue != null) {
                newVariable = new StringType(name, isArray, assignmentvalue);
            } else {
                newVariable = new StringType(name, isArray);
            }
            break;
        case "char":
            if (name == null) {
                if (arraySign == null) {
                    // Creating a raw parameter
                    newVariable = new CharType();
                } else {
                    // Creating a return type variable for a method declaration
                    newVariable = new CharType(isArray);
                }
            } else if (assignmentvalue != null) {
                newVariable = new CharType(name, isArray, assignmentvalue);
            } else {
                newVariable = new CharType(name, isArray);
            }
            break;
        case "double":
            if (name == null) {
                if (arraySign == null) {
                    // Creating a raw parameter
                    newVariable = new DoubleType();
                } else {
                    // Creating a return type variable for a method declaration
                    newVariable = new DoubleType(isArray);
                }
            } else if (assignmentvalue != null) {
                newVariable = new DoubleType(name, isArray, assignmentvalue);
            } else {
                newVariable = new DoubleType(name, isArray);
            }
            break;
        default:
            throw new IllegalVariableTypeException("Variable type error");
        }
        return newVariable;
    }
}
```

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

```java
package oop.ex7.main.parameters.variables;

import oop.ex7.main.parameters.ParameterNotExistException;

/**
 * This exception indicates that a variable isn't exist, compilation error.
 *
 * @author roeia1
 *
 */
public class VariableNotExistException extends ParameterNotExistException {

    public VariableNotExistException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

43 oop/ex7/main/parameters/variables/VariableTypeM

```java
package oop.ex7.main.parameters.variables;

/**
 * This exception indicates that there is a variable mismatch to the assigned
 * variable.
 *
 * @author roeia1
 *
 */
public class VariableTypeMismatchException extends
    VariableCompilationException {

    public VariableTypeMismatchException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```

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

```java
package oop.ex7.main.parameters.variables;

/**
 * This exception indicates that there is a method calling returning void used
 * in assignment.
 *
 * @author roeia1
 *
 */
public class VoidMethodException extends VariableCompilationException {

    public VoidMethodException(String errorMessage) {
        super(errorMessage);
    }

    /**
     *
     */
    private static final long serialVersionUID = 1L;

}
```