Lab1
Introduction to the EO language

1.Aims of the Lab1.

1. To introduce EO language
2. Set up the environment for development
3. Write and Run “Hello world!”

2. Guidelines

1) Lab1 is aimed to give the first view of EO language
2) To set up an environment for development using EO
3) Run “Hello world!”

3. Theoretical materials

EO (stands for Elegant Objects or ISO 639-1 code of Esperanto) is an object-oriented programming language. It is still a prototype and is the future of OOP [5]. EO is one of the promising technologies that arise to drive the course of elaboration on the proper practical application of the OOP paradigm. The EO philosophy advocates the concept of so-called Elegant Objects, thus, pure objects free from the incorrectly taken design decisions common to the mainstream technologies. Specifically, these are: Static methods and attributes; Classes; Implementation inheritance; Mutable objects; Null references; Global variables and methods; Reflection and annotations; Typecasting; Scalar data types; Flow control operators (for loop, while loop, etc.).

Eolang is an object-oriented programming language aimed at realizing the pure concept of object-oriented programming, in which all components of a program are objects. Eolang's main goal is to prove that fully object-oriented programming is possible not only in books and abstract examples but also in real program code aimed at solving practical problems.

```
program     ::= [ license ] [ metas ] objects
objects     ::= object EOL { object EOL }
license     ::= { comment } EOL
metas       ::= { meta } EOL
comment     ::= '#' { ANY } EOL
meta        ::= '+' name [ '␣' ANY { ANY } ] EOL
name        ::= /[a-z]/ { ANY }
object      ::= ( foreign — local )
foreign     ::= abstraction '␣ /' name
local       ::= ( abstraction — application ) details
details     ::= [ tail ] { vtail }
tail        ::= EOL TAB { object EOL } UNTAB
vtail       ::= EOL ref [ htail ] [ suffix ] [ tail ]
abstraction ::= attributes [ suffix ]
attributes  ::= '[' attribute { '␣' attribute } ']'
attribute   ::= '@' — name [ '...' ]
suffix      ::= '␣' '>' '␣' ( '@' — name ) [ '!' ]
ref         ::= '.' ( name — '^' )
application ::= head [ htail ]
htail       ::= application ref
            — '(' application ')'
            — application ':' name
            — application suffix
            — application '␣' application
head        ::= name — data — '@' — '$'
            — '^' — '*' — name '.'
data        ::= bytes — string — integer
            — char — float — regex
bytes       ::= byte { '-' byte }
byte        ::= /[\dA-F][\dA-F]/
string      ::= /"[^"]*"/
integer     ::= /[+-]?\d+|0x[a-f\d]+/
char        ::= /'([^']|\\\d+)'/
regex       ::= //.+/[a-z]*/
float       ::= /[+-]?\d+(\.\d+)?/ [ exp ]
exp         ::= /e(+|-)?\d+/
```

Figure 1. The full syntax of EO in BNF. EOL is a line ending that preserves the indentation of the previous line. TAB is a right-shift of the indentation, while UNTAB is a left-shift. ANY is any symbol excluding EOL and '␣'. The texts between forward slashes are Perl-style regular expressions.

How to add and run “Hello world!”

EO core repository is located here[https://github.com/cqfn/eo].

First, clone this repo to your local machine and go to the `eo` directory (you will need Git installed):

```
$ git clone https://github.com/cqfn/eo.git
$ cd eo
```

Second, compile the transpiler and the runtime of the EO programming language (you will need Maven 3.3+ and Java SDK 8+ installed):

```
$ mvn clean install
```

You need to run the above command only once. This will install the runtime & the transpiler to your machine.
Then, compile the code of the sandbox:

```
$ cd sandbox/hse
$ mvn clean compile
```

Intermediary `*.xml` files will be generated in the `target` directory (it will be created). Also, there will be `*.java` and `*.class` files. Feel free to analyze them: EO is parsed into XML, then translated to Java, and then compiled by Java SDK to Java bytecode. Finally, just run the bytecode program through JRE. For example, to run the Fibonacci example, do the following:

```
$ ./run.sh appFibonacci 9
The program has dataized to: 9th Fibonacci number is 34


real    0m0.177s
user    0m0.175s
sys     0m0.037s
```

1. Create new file lab1.eo in sandbox folder,and enter the code bellow:

```
+alias stdout org.eolang.io.stdout
```

```
[] > lab1
  stdout > @
    "Hello, world!"
```

It is **important** to remember that each EO program **must** be ended with a new line without any symbols.

Compile and run your program.

4. Order of work

- examine the material from section 3
- set up/check the actual version of Maven and Java(you will need Maven 3.3+ and Java SDK 8+)
- clone the repository to the local machine
- compile the transpiler and the runtime
- compile sandbox existing code
- write “Hello world!” program

5. Self-practice

1. Create and run “Hello, world!”
2. Modify “Hello, world!”, feel free to enter any text.

6. Control questions

1) What is EO language according to its philosophy?
2) What problem EO could potentially solve?
3) Is EO open-source language?