Lab2 First operations

1.Aims of the Lab2.

1. To introduce EO operations
2. To develop simple mathematical examples

2. Guidelines

1) Lab2 is aimed to introduce EO operations and objects
2) To develop 5 mathematical solutions

3. Theoretical materials

Objects

Objects are a centric notion of the EO programming language. Essentially, an object is a set of attributes. An object connects with and links other objects through its attributes to compose a new concept that the object abstracts.
An abstract object is an object that has at least one free attribute. This is an example of an abstract object:

```
[a b] > sum
  a.add b > @
  a > leftOperand
  b > rightOperand
```

A closed object is an object whose all attributes are bound. These are examples of closed objects:

```
# Application can turn an abstract object to a closed one
sum 2 5 > closedCopyOfSum
# Abstraction can declare closed objects
[] > zero
  0 > @
  "0" > stringValue
  # Closed objects may have abstract attributes
  [x] > add
    sum 0 x > @
  # And closed attributes, too
  [] > neg
    -0 > @
  $.add 1 > addOne
```

Attributes

An attribute is a pair of a name and a value, where a value of an attribute is another object. That is because "Everything in EO is an object". Hence, for instance, an attribute name of an object person may be also referred to as plainly the object name of the object person.

Free & Bound Attributes. Binding

Binding is an operation of associating an attribute's value with some object. An attribute may be bound to some object only once. An attribute that is not bound to any object is named a free attribute. An attribute that has some object associated with its value is called a bound attribute. Free attributes may be declared through the object abstraction only. Binding may be performed either during object declaration using the bind (>) operator (see the abstraction section for more information) or through object copying (see the application section for details).

Accessing Attributes. The Dot Notation

There are no access modifiers in the EO programming language. All attributes of all objects are publicly visible and accessible. To access attributes of objects, the dot notation is used. The dot notation can be used to retrieve values of attributes and not to bind attributes with objects.

Example. The Horizontal Dot Notation

```
(5.add 7).mul 10 > calc
```

Example. The Vertical Dot Notation

```
mul. > calc
  add.
    5
    7
  10
```

Here, add is an attribute of the object 5 and mul is an attribute of the attribute object add (or, more precisely, an attribute of an object that add abstracts or dataizes to, which is an integer number int).

The @ attribute

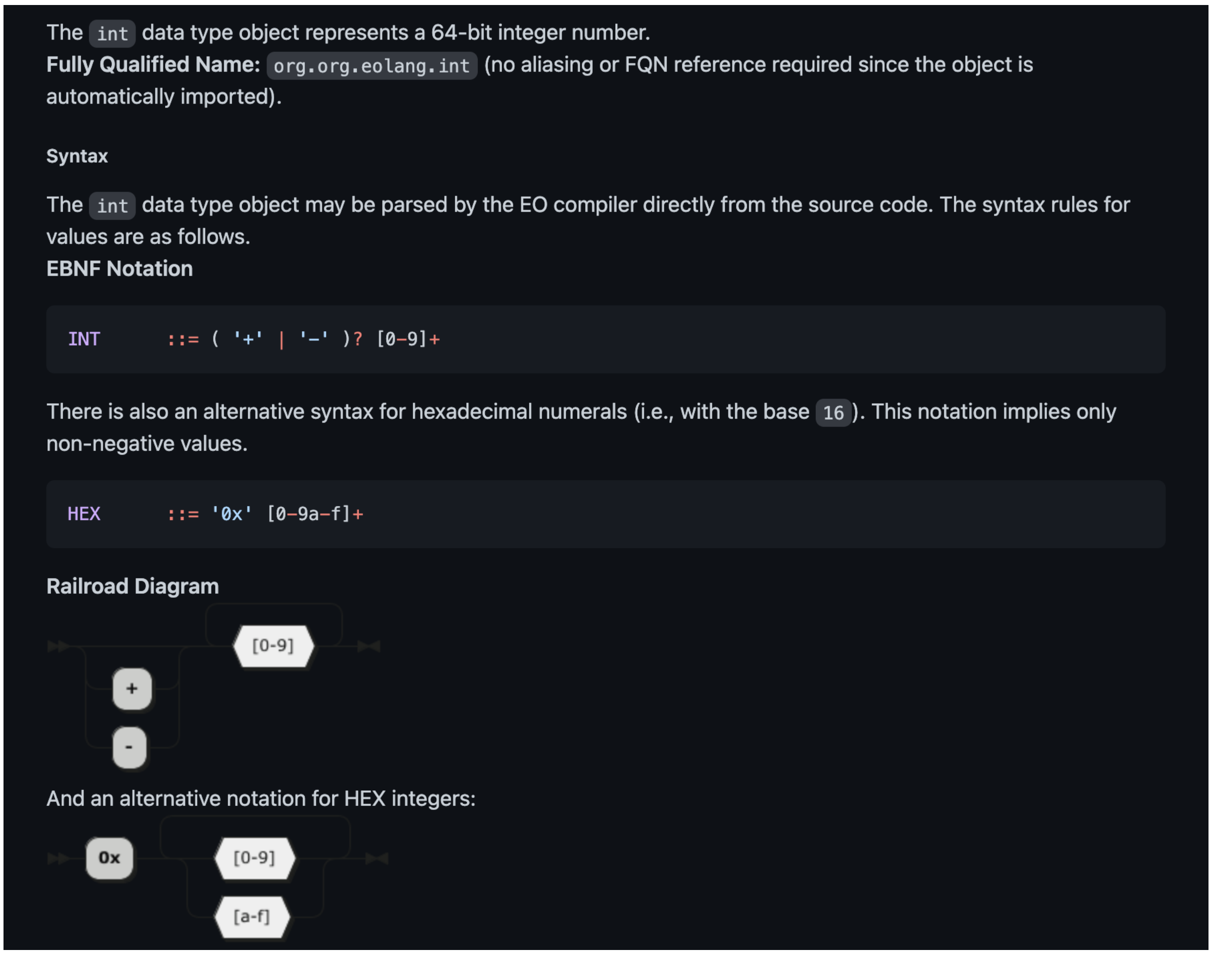
The @ attribute is named phi (after the Greek letter φ). The @ character is reserved for the phi attribute and cannot be used for any other purpose. Every object has its own and only @ attribute. The @ attribute can be bound to a value only once. The @ attribute is used for decorating objects. An object bound to the @ attribute is referred to as a decoratee (i.e., an object that is being decorated) while the base object of the @ attribute is a decorator (i.e., an object that decorates the decoratee). Since the @ attribute may be bound only once, every object may have only one decoratee object. More on the decoration see in this section. Besides, the @ attribute is heavily used in the dataization process.

The ^ attribute

The ^ attribute is used to refer to the parent object. The ^ attribute may be used to access attributes of a parent object inside of the current object with the dot notation (e.g., ^.attrA).

Example
```
[] > parentObject
  42 > magicNumbe
  [] > childObject
    24 > magicNumber
    add. > @
      ^.magicNumber # refers to the parent object's attr
      magicNumber # refers to $.magicNumber
```

int Data Type Object



The `int` data type object represents a 64-bit integer number.
Fully Qualified Name: `org.org.eolang.int` (no aliasing or FQN reference required since the object is automatically imported).

Syntax

The `int` data type object may be parsed by the EO compiler directly from the source code. The syntax rules for values are as follows.
EBNF Notation

```
INT      ::= ( '+' | '-' )? [0-9]+
```

There is also an alternative syntax for hexadecimal numerals (i.e., with the base `16`). This notation implies only non-negative values.

```
HEX      ::= '0x' [0-9a-f]+
```

Railroad Diagram

And an alternative notation for HEX integers:

eq Attribute

The eq attribute object is used for testing if two int objects are equal.
The eq attribute object has one free attribute x of type int that is the second object (the first object is the base object of the eq attribute).
If the eq attribute object is fully applied, it represents the result of the equality testing (either true (if the objects are equal) or false (otherwise)).
Example

```
+package sandbox
+alias sprintf org.org.eolang.txt.sprintf
+alias stdout org.org.eolang.io.stdout

[args...] > app
  stdout > @
    sprintf
      "%b\n%b\n"
      eq.
        0xf
        15
      15.eq (0xf.add 1)
```

Running

```
IN$: ./run.sh
OUT>: true
OUT>: false
IN$:
```

less Attribute

The less attribute object is used for testing if its base int object is less than its x free attribute (i.e. $ < x).
If the less attribute object is fully applied, it represents the result of the testing (either true (if the base object is less than x free attribute of the less) or false (otherwise)).
Example

```
+package sandbox
+alias sprintf org.org.eolang.txt.sprintf
+alias stdout org.org.eolang.io.stdout

[args...] > app
  stdout > @
    sprintf
      "%b\n%b\n"
      -7.less 0
      less.
        0
        0
```

Running
```
IN$: ./run.sh
OUT>: true
OUT>: false
IN$:
```

add Attribute

The add attribute object is used to calculate the sum of its base int object and the free attribute x of type int (i.e. $+x).
If the add attribute object is fully applied, it represents the resulting sum of the integer numbers.
Example
```
+package sandbox
+alias sprintf org.org.eolang.txt.sprintf
+alias stdout org.org.eolang.io.stdout

[args...] > app
  stdout > @
    sprintf
      "%d\n%d\n"
      add.
        0x10
        16
      -16.add 0x10
```

Running
```
IN$: ./run.sh
OUT>: 32
OUT>: 0
IN$:
```

sub Attribute

The sub attribute object is used to calculate the difference between its base int object and the free attribute x of type int (i.e. $-x).
If the sub attribute object is fully applied, it represents the resulting difference of the integer numbers.
Example
```
+package sandbox
+alias sprintf org.org.eolang.txt.sprintf
+alias stdout org.org.eolang.io.stdout
```

```
[args...] > app
  stdout > @
    sprintf
      "%d\n%d\n"
      sub.
        0x10
        16
      -16.sub 0x10
```

Running

```
IN$: ./run.sh
OUT>: 0
OUT>: -32
IN$:
```

neg Attribute

The neg attribute object is used to negate its base int object (i.e. -$).
If the neg attribute object is applied (called), it represents the resulting negation of the base int object.
Example

```
+package sandbox
+alias sprintf org.org.eolang.txt.sprintf
+alias stdout org.org.eolang.io.stdout

[args...] > app
  stdout > @
    sprintf
      "%d\n%d\n%d\n%d\n"
      5.neg
      0x10.neg
      (17.add 3).neg
      17.neg.add 3
```

Running

```
IN$: ./run.sh
OUT>: -5
OUT>: -16
OUT>: -20
OUT>: -14
IN$:
```

mul Attribute

The mul attribute object is used to calculate the product of its base int object and the free attribute x of type int (i.e. $ × x).
If the mul attribute object is fully applied, it represents the resulting product of the integer numbers.
Example

```
+package sandbox
+alias sprintf org.org.eolang.txt.sprintf
+alias stdout org.org.eolang.io.stdout

[args...] > app
  stdout > @
    sprintf
      "%d\n%d\n%d\n%d\n%d\n"
      -7.mul 0
      13.mul 1
      mul.
        0x10
        0x10
      ((10.mul 10).mul 10).mul 10
      10.mul 10.mul 10.mul 10
```

Running

```
IN$: ./run.sh
OUT>: 0
OUT>: 13
OUT>: 256
OUT>: 10000
OUT>: 10000
IN$:
```

mod Attribute

The mod attribute object is used to calculate the floor remainder of the integer division of its base int object by the x free attribute (i.e. $ fmod x).
If the mod attribute object is fully applied, it represents the resulting floor modulus (remainder).
The modulus for x = 0 is undefined. The resulting floor modulus has the same sign as the divisor x.
The relationship between the mod and div operations is as follows:
(x div y) * y + x mod y == x
Example

```
+package sandbox
+alias sprintf org.org.eolang.txt.sprintf
```

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

[args...] > app
  stdout > @
    sprintf
      "%d\n%d\n%d\n%d\n%d\n%d\n"
      2.mod 1
      7.mod 5
      113.mod 10
      113.mod -10
      -113.mod 10
      -113.mod -10
```

Running

```
IN$: ./run.sh
OUT>: 0
OUT>: 2
OUT>: 3
OUT>: -7
OUT>: 7
OUT>: -3
IN$:
```

pow Attribute

The pow attribute object is used to calculate the power of its base int object and the free attribute x of type int (i.e. $^x).
If the pow attribute object is fully applied, it represents the resulting power of the base int object raised to the power of the x attribute.
Example

```
+package sandbox
+alias sprintf org.org.eolang.txt.sprintf
+alias stdout org.org.eolang.io.stdout

[args...] > app
  stdout > @
    sprintf
      "%d\n%d\n%d\n%d\n%d\n"
      2.pow 10
      -2.pow 3
      2.pow -10
      2.pow 0
      2.pow 1
```

Running

```
IN$: ./run.sh
OUT>: 1024
OUT>: -8
OUT>: 0
OUT>: 1
OUT>: 2
IN$:
```

Here, 2^(-10) results in 0 as well as raising all the integer numbers (except 0) to the negative power (-1, -2, -3, ...).

Command Line Interface Output

The EO Standard Object Collection contains two objects for the CLI output: sprintf for strings formatting and stdout for plain text output.

Plain Text Output. stdout

For plain text output, the stdout object is used.
Fully Qualified Name: org.org.eolang.io.stdout.

Usage

The stdout object has one free attribute text that should be bound to the text to print.
The object bound to the text attribute must be of string type.
The stdout does not put the End of Line character at the end of the output, so the \n escape sequence should be used in case if such a behavior is needed.
For the complete list of escape sequences supported by stdout, see the corresponding section of the article.

Example 1. The Plain Old “Hello, World”

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

[args...] > app
  (stdout "Hello, World!\n") > @
```

Running

```
IN$: ./run.sh
OUT>: Hello, World!
IN$:
```

Example 2. Print the First Word of the User's Input

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

```
[args...] > app
  stdout > @
    get.
      args
      0
```

Running

```
IN$: ./run.sh Hello Bye Thanks Ok
OUT>: HelloIN$:
```

Note: here, the Hello is printed with no EOL character at the end of the line because of the absence of it in the user input.

Formatting Strings. sprintf

For strings formatting, the sprintf object is used.
String formatting is the process of data injection into the string, optionally applying format patterns to the data.
Fully Qualified Name: org.org.eolang.txt.sprintf.

Usage

The sprintf object has two free attributes:

1. format for the format string that describes the formatting of the resulting string.
2. args for the data being injected into the string. args may be empty or may have any number of objects. args must be consistent with the format (i.e., the number and the types (as well as their order) of the objects in the format and the args should be the same).

If the sprintf object is fully applied, it represents the resulting formatted string.
For the format syntax reference, see this article.

Example. Format 'Em All

```
+package sandbox
+alias sprintf org.org.eolang.txt.sprintf
+alias stdout org.org.eolang.io.stdout

[args...] > app
  sprintf > formatted_string
    "int: %d, bool: %b, string: %s\n"
    2
    (2.less 0)
    "Hey"

  (stdout formatted_string) > @
```

Running

```
IN$: ./run.sh
OUT>: int: 2, bool: false, string: Hey
IN$:
```

4. Order of work

- examine the material from section 3
- develop 5 eo programs from section self-practice

5. Self-practice

Below is the list of mathematical tasks, choose any 5 and implement them(in one/five solutions)

1. 256+7
2. 763+2875
3. 854722+753728
4. 5637173831+547428139
5. 75*2
6. 546465*2139593
7. 127495*348129
8. 13713805*98461
9. -(40)
10. -(70)
11. -(246475)
12. -(24452317)
13.
14. -(-(356137))
15. (5+7)=?(6*2)
16. (7*8)=?(41+7)
17. (3131+74136)=?(62*243)
18. (256+16)=?(139*2)
19. 5<8
20. 212371>129457
21. 4781264<32853719
22. 5^7
23. 4^5
24. 641^82
25. 5679^4

6. Control questions

1) Objects in EO are?
2) How to output the program result?
3) What does pow operation?
4) What means “@” attribute?
5) How to format Strings in EO?