OOP 2nd Assignment - Documentation

ALI AFZAL **2. assignment/4th task** 28th April 2022

BLVHI9
blvhi9@inf.elte.hu
Group 1

TASK

4. Layers of gases are given, with certain type (ozone, oxygen, carbon dioxide) and thickness, affected by atmospheric variables (thunderstorm, sunshine, other effects). When a part of one layer changes into another layer due to an athmospheric variable, the newly transformed layer ascends and engrosses the first identical type of layer of gases over it. In case there is no identical layer above, it creates a new layer on the top of the atmosphere.
In the following we declare, how the different types of layers react to the different variables by changing their type and thickness.
No layer can have a thickness less than 0.5 km, unless it ascends to the identical-type upper layer. In case there is no identical one, the layer perishes.

	thunderstorm	sunshine	other
ozone	-	-	5% turns to oxygen
oxygen	50% turns to ozone	5% turns to ozone	10% turns to carbon di-oxide
carbon dioxide	-	5% turns to oxygen	-

The program reads data from a text file. The first line of the file contains a single integer N indicating the number of layers. Each of the following N lines contains the attributes of a layer separated by spaces: type and thickness. The type is identified by a character: Z – ozone, X – oxygen, C – carbon dioxide.
The last line of the file represents the atmospheric variables in the form of a sequence of characters: T – thunderstorm, S – sunshine, O – others. In case the simulation is over, it continues from the beginning.

The program should continue the simulation until the number of layers is the triple of the initial number of layers or is less than three. The program should print all attributes of the layers by simulation rounds!

The program should ask for a filename, then print the content of the input file. You can assume that the input file is correct. Sample input:

```
4
Z 5
X 0.8
C 3
X 4
OOOOSSTSSOO
```

Plan

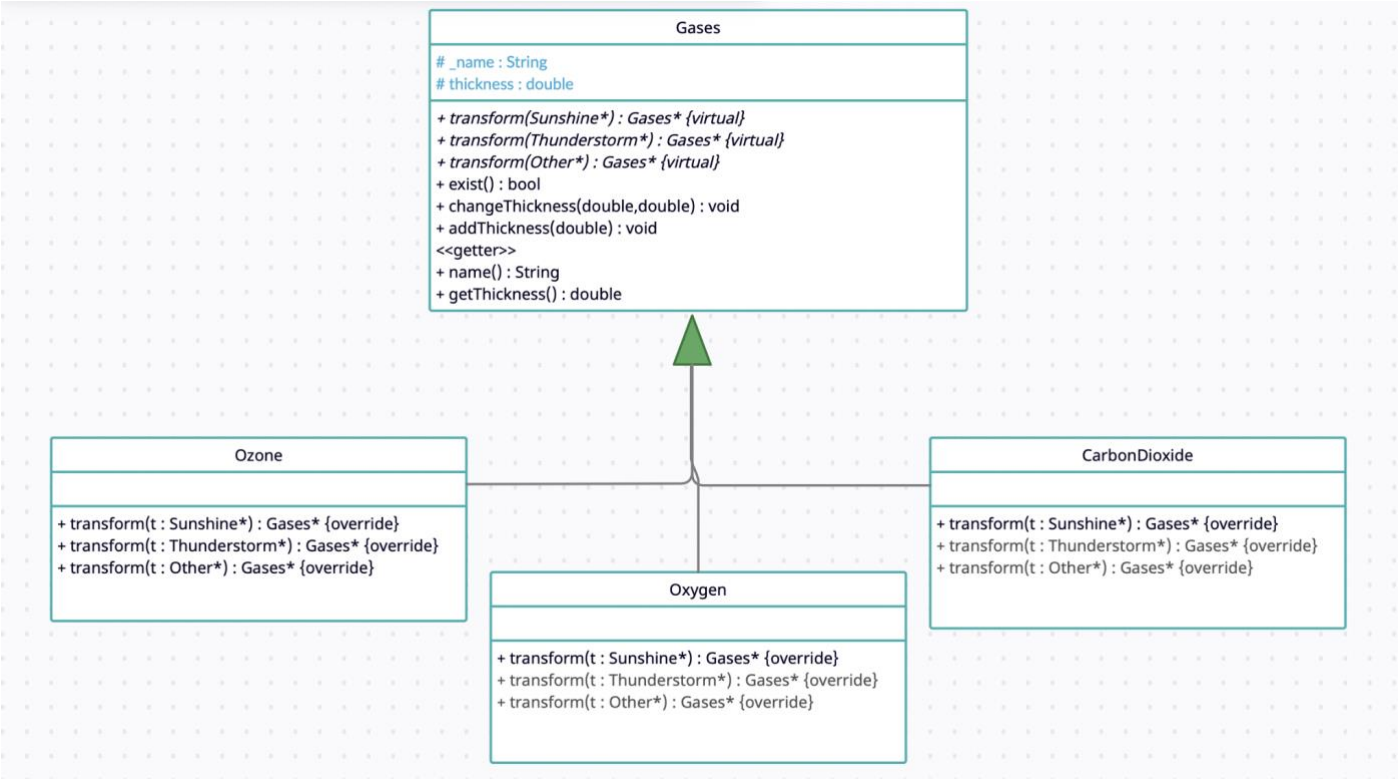
To describe the Atmospheric Gases, 4 classes are introduced: base class *Gases* to describe the general properties and 3 children for the concrete gas: *Ozone*, *Oxygen*, and *CarbonDioxide*. Regardless the type of the gases, they have several common properties, like the name (*_name*) and the thickness (*thickness*), the getter of its name (*name()*), getter of its thickness (*getThickness()*), if it exists (*exist()*) and it can be examined what happens when an atmospheric variable acts on it. Every concrete gas has three methods (*transform()*) that show how a *Thunderstorm*, a *Sunshine*, or *Other* changes the gas. This latter operation (*transform()*) modifies the thickness of the Gases and transform it. Operations *exist()* and *name()* may be implemented in the base class already, but *transform()* just on the level of the concrete classes as its effect depends on the type of the gases. Therefore, the general class *Gases* is going to be abstract, as method *transform()* is abstract and we do not wish to instantiate such class.

General description of the atmospheric variable is done in the base class *Conditions* from which concrete variable are inherited: *Thunderstorm*, *Sunshine*, and *Other*. Every concrete variable has a method that show how a *Gases* changes when it acts on it. Objects are referred by pointers.

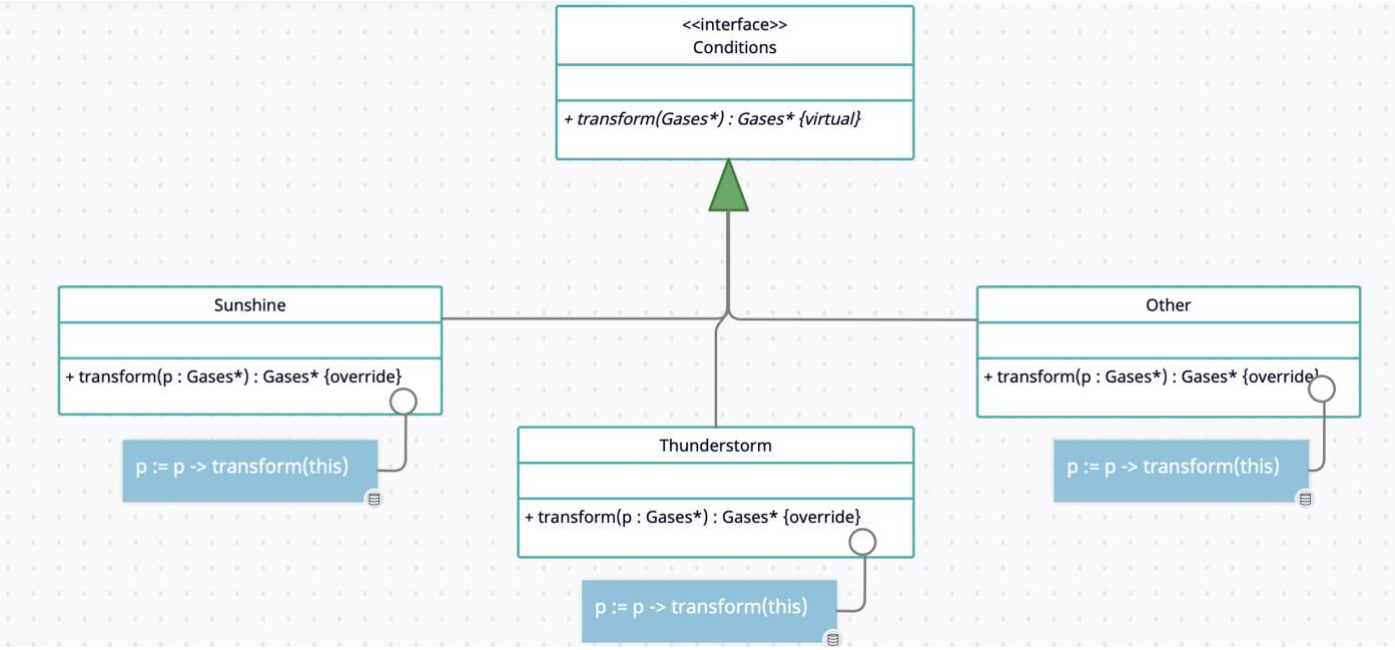
The special *Gases* classes initialize the name and the thickness through the constructor of the base class and override the operation *transform()* in a unique way. According to the following table, in method *transform()*, conditionals have to be used in which the type of the Atmospheric *Conditions* is examined. Though, the conditionals are not effective if the program might be extended by new atmospheric variable types, as all of the methods *transform()* in all of the concrete creature classes have to be modified. To avoid it, design pattern Visitor is applied where the ground *Gases* are going to have the role of the visitor.

	thunderstorm	sunshine	other
ozone	-	-	5% turns to oxygen
oxygen	50% turns to ozone	5% turns to ozone	10% turns to carbon di-oxide
carbon dioxide	-	5% turns to oxygen	-

UML Class Diagram:-



Methods *transform()* of the concrete Gases expect a Conditions object as an input parameter as a visitor and calls the methods which corresponds to the species of the Gases.



All the classes of the Conditions are realized based on the Singleton design pattern, as it is enough to create one object for each class.

Specification:-

A = **gases : Gases^{n}, agent : Conditions^{m}, ans : Gases*Gases**

Pre = **gases = gases$_0$ ∧ *agent* = *agent$_0$***

Post = **agent = agent$_n$ ∧**

∀i ∈[1..m]: ∀j ∈[1..n]: (Gases temp := agent[i].transform(gases[j])) ∧

!(gases[j+1] = temp) ∧ gases[n+1] := temp ∧ ans[1..] =$\bigoplus_{i=1..n}$ < gases >

|gases| >=3 && |gases| < (3*n)

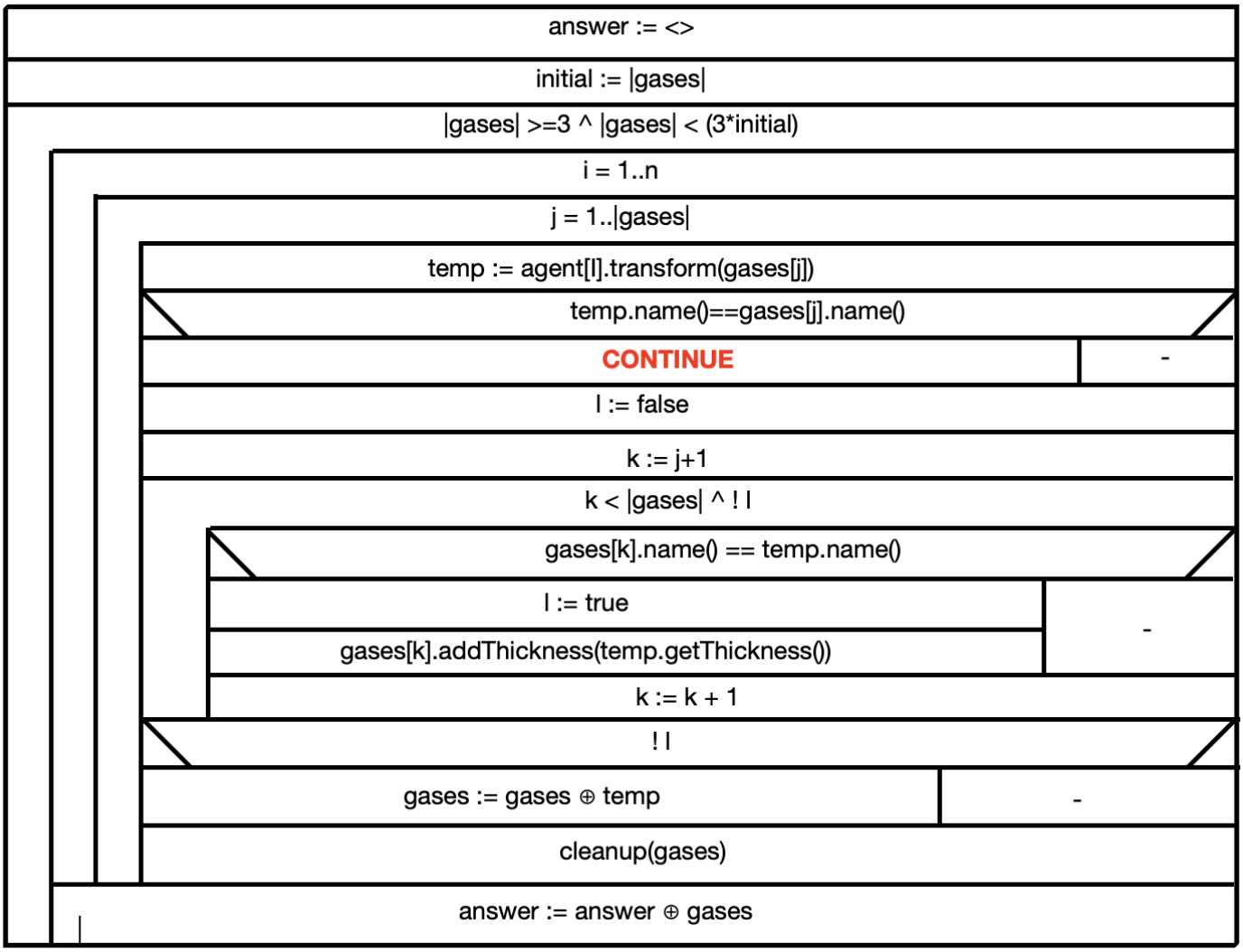
Only two **predefined** algorithm patterns are used:

Analogy with two Conditional Summation:

enor(E)	***i = j+1..|gases|***
f(e)	***gases[j] = temp***
s	***gases[j].getThickness***
H,+,0	***R,+,0.0***

enor(E)	***i = 0..|gases|***
f(e)	***gases[i] != temp***
s	***ans***
H,+,0	***Gases*,⊕,<>***

Algorithm:-



Testing:-

Grey box test cases:

1) Outer Loop
   a) zero condition
   b) one condition
   c) more conditions
   d) first conditions simulation transforms the Gases properly depending on the type
   e) last conditions simulation transforms the Gases properly depending on the type

2) Inner Loop
   a) zero gas
   b) one gas
   c) more gases
   d) first gas properly transformed depending on the type of conditions
   e) last gas properly transformed depending on the type of conditions

3) Examination of function *transform()*

Different cases depending on the Gases and the Conditions.