OOP Sreang Vivattanak: Task 1, 1st assignment.

Sreang Vivattanak
H2G7BJ
h2g7bj@inf.elte.hu
Group 3

2nd.assignment/ task 3

9th May 2024

Task 3

Create a program to simulate the effects of weather conditions on different layers of gas in the atmosphere.

3. Layers of gases are given, with certain types (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 atmospheric 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 one gas component totally perishes from the atmosphere. 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:

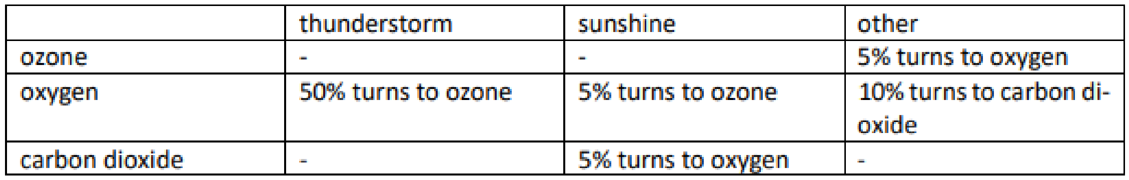
```
4
Z 5
X 0.8
C 3
X 4
OOOOSSTSSOO
```

Plan

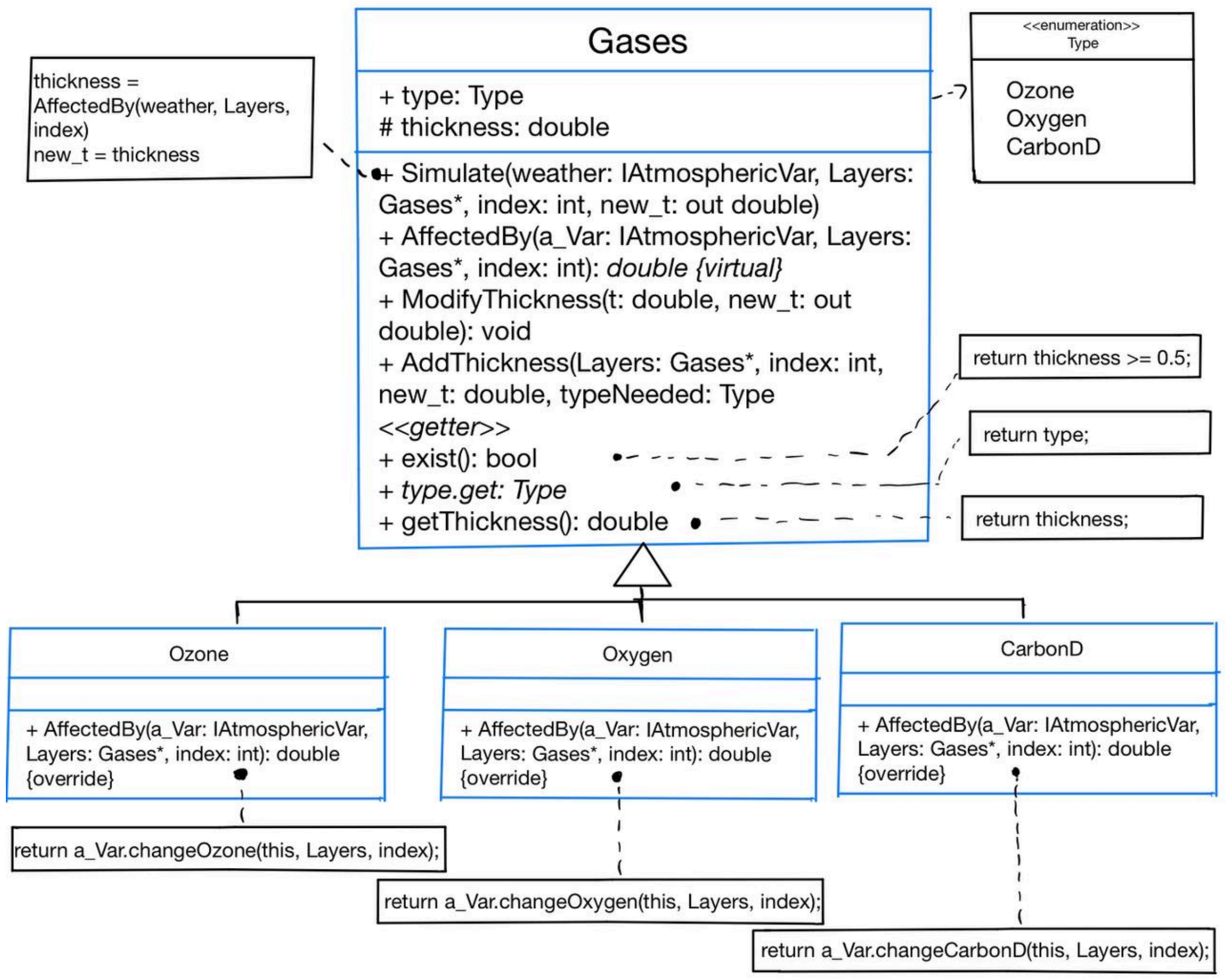
To describe the atmospheric gases, 4 classes are introduced: base class *Gases* to describe the general properties and 3 children for the concrete gases: *Ozone, Oxygen,* and *Carbon Dioxide*. Regardless of the type of gases, they have several common properties, like the Type(*type*), and the thickness(*thickness*), if it exists(*Exist()*), getter of its thickness(*GetThickness()*), and Type(*type.get*). It also has 2 other methods to modify the thickness(*ModifyThickness()*), and to add the thickness/create new gas(*AddThickness()*). So, it can be examined when the gases are affected by the atmospheric variable. This latter operation (*AffectedBy()*), reduces the thickness of a gas, and then adds/creates a new gas with the thickness reduced. Operations *Exist()*, *GetThickness()*, *type.get*, *ModifyThickness()*, *AddThickness()*, may be implemented in the base class already, but *AffectedBy()* just on the level of concrete classes as its effect depends on the type of gas. Therefore, the general class *Gases* is going to be abstract, as the method *AffectedBy()* is abstract and we do not wish to instantiate such a class.

General description of the atmospheric variable is done in the base class *AtmosphericVar* from which concrete variables are inherited: *Thunderstorm, Sunshine,* and *Other*. Every concrete variable has three methods that show how an Ozone, an Oxygen, or a Carbon Dioxide changes when it is affected by it, but also sometimes create a new gas layer/ add thickness to a layer above.

The special *Gases* classes initialize its type, and thickness through the constructor of the base class and overrides the operation *AffectedBy()* in an unique way. According to the table below, in the method *AffectedBy(),* conditionals have to be used in which the type of *AtmosphericVar* is examined. Though, the conditionals would violate the SOLID principle of OOP, and it might not be effective if the program might be extended by new *AtmosphericVar* type, as all of the methods *AffectedBy()* in all the concrete gas classes have to be modified. To avoid it, the design pattern Visitor is applied where the ground *Gases* are going to have the role of the visitor.

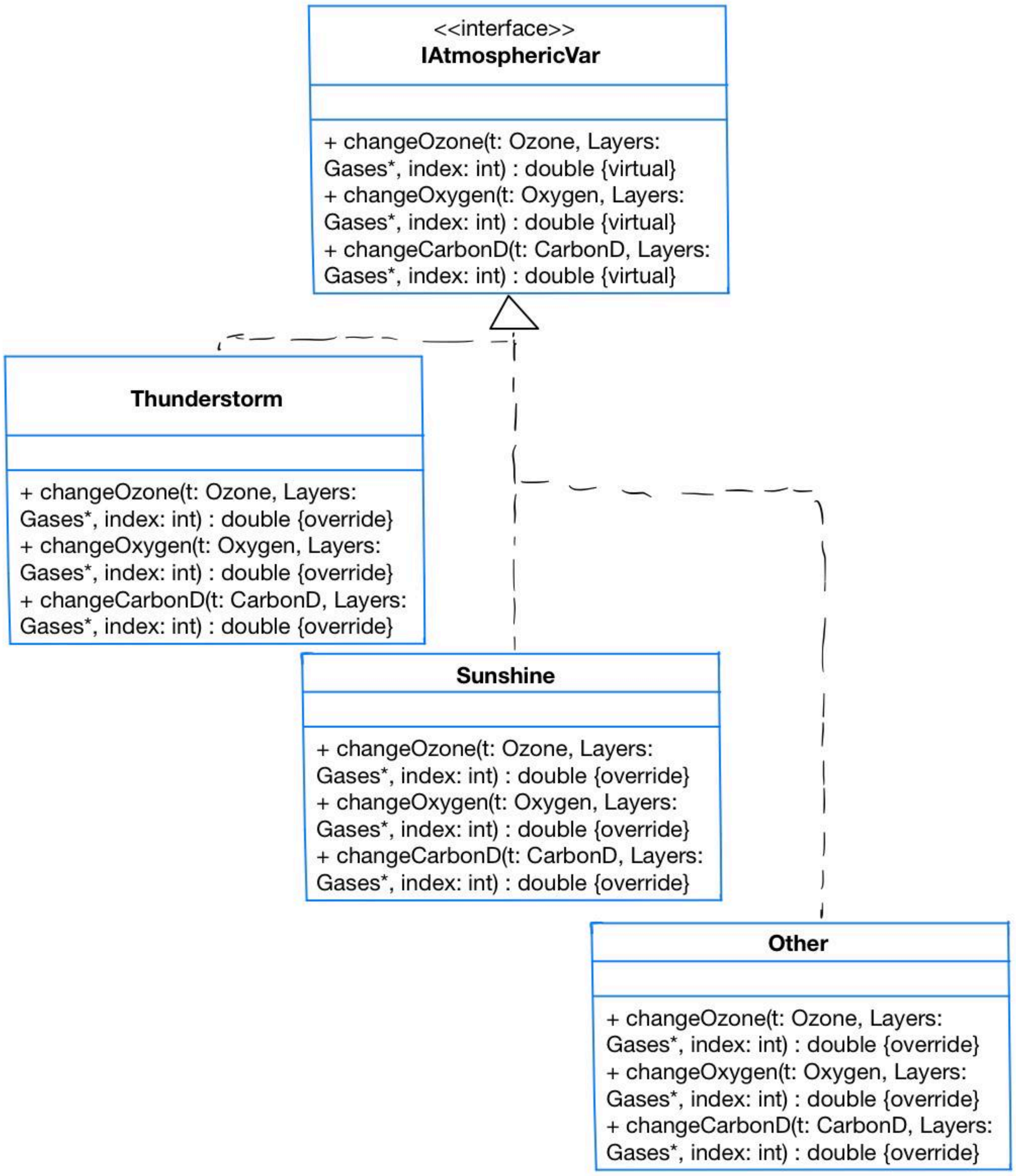


	thunderstorm	sunshine	other
ozone	-	-	5% turns to oxygen
oxygen	50% turns to ozone	5% turns to ozone	10% turns to carbon di-oxide
carbon dioxide	-	5% turns to oxygen	-



Gases
+ type: Type
# thickness: double
+ Simulate(weather: IAtmosphericVar, Layers: Gases*, index: int, new_t: out double)
+ AffectedBy(a_Var: IAtmosphericVar, Layers: Gases*, index: int): double {virtual}
+ ModifyThickness(t: double, new_t: out double): void
+ AddThickness(Layers: Gases*, index: int, new_t: double, typeNeeded: Type
<<getter>>
+ exist(): bool
+ type.get: Type
+ getThickness(): double
thickness = AffectedBy(weather, Layers, index)
new_t = thickness
<<enumeration>>
Type
Ozone
Oxygen
CarbonD
return thickness >= 0.5;
return type;
return thickness;
Ozone
+ AffectedBy(a_Var: IAtmosphericVar, Layers: Gases*, index: int): double {override}
Oxygen
+ AffectedBy(a_Var: IAtmosphericVar, Layers: Gases*, index: int): double {override}
CarbonD
+ AffectedBy(a_Var: IAtmosphericVar, Layers: Gases*, index: int): double {override}
return a_Var.changeOzone(this, Layers, index);
return a_Var.changeOxygen(this, Layers, index);
return a_Var.changeCarbonD(this, Layers, index);

Methods *AffectedBy()* of the concrete *Gases* expect a *IAtmosphericVar* object as an input parameter as a visitor and calls the methods which corresponds to the type of *Gases*. The Layers and index are for the other methods not related to the Visitor design pattern.



<<interface>>
IAtmosphericVar
+ changeOzone(t: Ozone, Layers: Gases*, index: int) : double {virtual}
+ changeOxygen(t: Oxygen, Layers: Gases*, index: int) : double {virtual}
+ changeCarbonD(t: CarbonD, Layers: Gases*, index: int) : double {virtual}
Thunderstorm
+ changeOzone(t: Ozone, Layers: Gases*, index: int) : double {override}
+ changeOxygen(t: Oxygen, Layers: Gases*, index: int) : double {override}
+ changeCarbonD(t: CarbonD, Layers: Gases*, index: int) : double {override}
Sunshine
+ changeOzone(t: Ozone, Layers: Gases*, index: int) : double {override}
+ changeOxygen(t: Oxygen, Layers: Gases*, index: int) : double {override}
+ changeCarbonD(t: CarbonD, Layers: Gases*, index: int) : double {override}
Other
+ changeOzone(t: Ozone, Layers: Gases*, index: int) : double {override}
+ changeOxygen(t: Oxygen, Layers: Gases*, index: int) : double {override}
+ changeCarbonD(t: CarbonD, Layers: Gases*, index: int) : double {override}

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

Specification

In the specification, when 1 gas is affected by an atmospheric variable it is denoted by the function *AffectedBy:* $Gases \times AtmosphericVar^m -> Gases \times AtmosphericVar^m$, which sometimes give a new thickness for the gas, or change another gas as well in its list. i^{th} version of the atmospheric variable is denoted by *weather*, which the program is not going to be shown, it is going to be a temporary variable of the atmospheric variable, in which the program will iterate over again, and again until there is one less type of gas in the atmosphere.

A = $(Layers: Gases^n, weather: IAtmosphericVar^m, exist: Gases^*)$

Pre = $(Layers = Layers_0 \wedge weather = weather_0)$

Post = (

$weather = weather_0 \wedge \forall i \in [initialUniqueGasCount == UniqueGasCount]$:

$\forall j \in [1..n]$: $(Layers[j].Simulate(weather[i], ref\ Layers, ind, out\ new\_t) \wedge$

if (initialLayerCount != Layers.Count) then initialLayerCount =Layers.Count; j+=2;)

$\wedge\ exist[1..] = \bigoplus_{i=1..n} < Layers[j] >)$

(Layers[j].Exist())

For the initialUniqueGasCount, and UniqueGasCount we just get the length of the hashset. We use a hashmap to count the unique gases. Also, for the main program the *Simulate()*, we need to use 2 Summations.

Analogy:

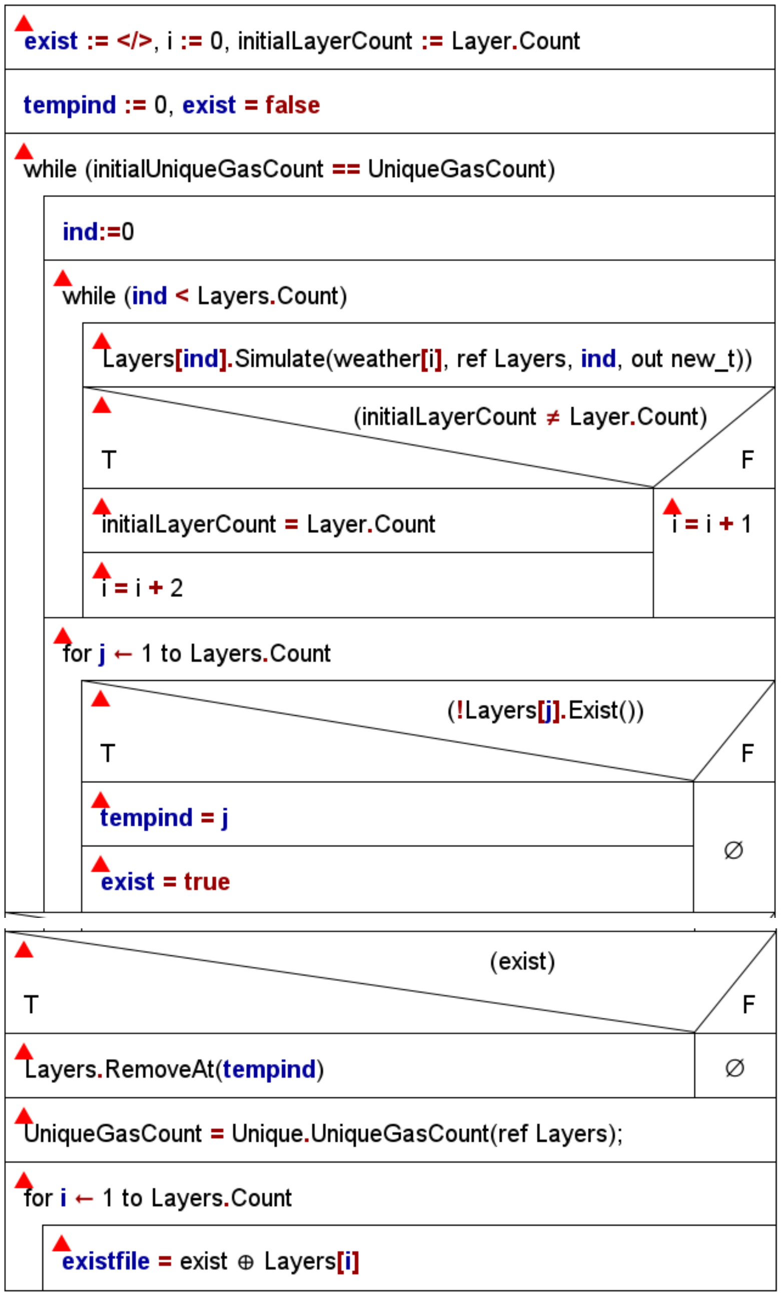
enor(E)	i = initialUniqueGasCount == UniqueGasCount
f(e)	Simulate(weather[i], ref Layers, ind, out new_t)
s	exist
H, +, 0	$Layers^*$, $\oplus$, <>

First component of the value of the function simulate().

enor(E)	j = 1…|Layers|
f(e)	Layers[j].GetThickness() = AffectedBy(weather[i], Layers, ind)
s	Layers[j].GetThickness()
H, +, 0	R, +, 0.0

Second component of the function simulates(), to change or add the thickness.

By merging the above loops, we can get the algorithm as such.



```
exist := </>, i := 0, initialLayerCount := Layer.Count
tempind := 0, exist = false
while (initialUniqueGasCount == UniqueGasCount)
    ind:=0
    while (ind < Layers.Count)
        Layers[ind].Simulate(weather[i], ref Layers, ind, out new_t))
        if (initialLayerCount ≠ Layer.Count)
            T:
                initialLayerCount = Layer.Count
                i = i + 2
            F:
                i = i + 1
    for j ← 1 to Layers.Count
        if (!Layers[j].Exist())
            T:
                tempind = j
                exist = true
            F:
                ∅
if (exist)
    T:
        Layers.RemoveAt(tempind)
    F:
        ∅
UniqueGasCount = Unique.UniqueGasCount(ref Layers);
for i ← 1 to Layers.Count
    existfile = exist ⊕ Layers[i]
```

We basically loop until (initialUniqueGasCount == UniqueGasCount) (we will define what these layers mean below), after this we will run the simulate function, making the gases in the layers all become *AffectedBy* the weather. We will adjust the thickness of each gas according to the problem they gave us.

Here is the specification to get the initialUniqueGasCount, and UniqueGasCount.

Specification:

A = (uniqueTypes : HashSet, l: int)
Pre = (uniqueTypes ∧ uniqueTypes0)
Post = $\forall i[1 ... Layers.Count]: uniqueType.Add(Layer.GetType()) \wedge \; l \; = \; |uniqueType|$

We run the initialUniqueGasCount at the beginning in order to get the initial value before we run any calculation on it. And we call this function at the end of the loop in order to update the UniqueGasCount to see if we need to continue looping.

Testing

Grey box test cases:

Outer Loop:

Length-based:

- zero atmospheric variable
- one atmospheric variable
- many atmospheric variable

Inner Loop:

Length-based:

- zero gas
- one gas
- more gases

First and Last:

- First gas properly transforming based on the type of atmospheric variable
- Last gas properly transforming based on the type of atmospheric variable

Examination of the function of Different methods:

- Input gas < 0.5, AddThickness, ModifyThickness, Simulate