LAYER OF GASES

B8crdq@inf.elte.hu	3rd./ Task	Second OOP Assignment
Talha Mujahid	Group 06	
B8crdq		

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 dioxide
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

Abstract Classes for Atmosphere and Gas Layers:

Create an abstract class Atmosphere with subclasses ThunderStorm, SunShine, and Other.Create an abstract class LayerOfGas to describe the general properties of gas layers with subclasses Ozone, Oxygen, and CarbonDioxide.

Singleton Pattern for Atmospheric Instances:

Implement the Singleton pattern for ThunderStorm, SunShine, and Other classes to ensure that only one instance of each can exist.

Gas Layer Operations:

Implement methods in the LayerOfGas class to handle gas layer properties and transformations:

type(): Returns the type of gas layer.

thickness(): Returns the thickness of the gas layer.

survived(): Checks if the gas layer has survived.

Traverse(): Modifies the thickness of the gas layer and changes the atmospheric variable.

Visitor Design Pattern:

Implement the Visitor design pattern where the atmospheric variable classes (ThunderStorm, SunShine, and Other) act as visitors.

Define methods in each atmospheric variable class to handle gas layer transformations:

Traverse() method in each atmospheric variable class to handle specific gas layer transformations without using conditionals.

Gas Layer Interaction and Modifications:

Implement methods in the Atmosphere class to handle interactions between gas layers and atmospheric variables:

ChangeGasLayer(): Modifies gas layers based on specified conditions and atmospheric variables.

Exception Handling:

LAYER OF GASES

Define custom exception class EmptyListException to handle empty lists in gas layer operations.

Main Program Execution:

Instantiate atmospheric variables and gas layers as needed.

Use the ChangeGasLayer() method to modify gas layers based on atmospheric conditions.

Handle exceptions if encountered during gas layer operations.

In the Specification, the atmospherics variables changes into one layer to another, which is denoted by the function Traverse : Layer x Atmposphere^m $\rightarrow$ Layer x Atmposphere^m which gives the changed layer or layer itself denoted whether it perished or not. Ith version of the Atmosphere Layer of the Atmosphere Layer Variables denoted by atmosphere Variables. Which the program does not show, and the End the When one gase perishes the Atmosphere gas stop.

A = Atmosphere Variable: Atmoshpere^m, layers : Gases Layer^n, mergeOrPerishedPerished: Layer Type

layerOfGases = rec(name: char, GetThickness: double)

Pre = layerOfGas = layers$_0$ $\wedge$ atmospheres = atmospheres$_0$

Post = atmosphere = atmosphere $\wedge$ $\forall$i$\in$[1…n].layer[i], atmosphere$_i$ = Transform(layers$_0$ [i],atmosphere$_{i-1}$) $\wedge$ perished = $\bigoplus_{i=1..n}$ < layer[i], type, layers[i].GetThickness >

Layer[i].perished()

In essence, the notation describes how a function or method transforms the state of atmospheric variables and their corresponding layers of gases, producing perished layers as a result. The pre-conditions define the initial state, while the post-conditions specify the final state after the operation.

Analogy:

Enor(E)	***i= 1…n***
f(e)	***Handler(layers[i],*** ***Atmosphere)1***
s	***LayerOfGas***
H,+,0	***LayerOfGas**** , ⊜, layers[i]

First component of the function Handler()

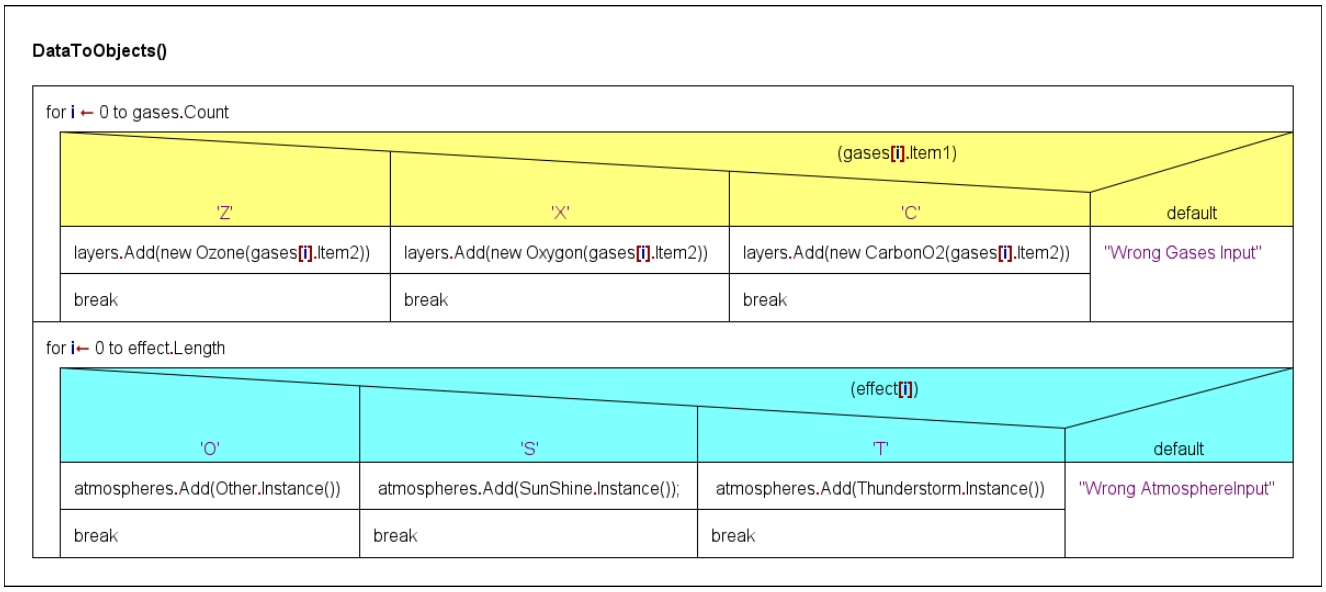
Handler of the Values of Function MergeOrPerished

Enor(E)	***i = 1..n***
f(e)	***MergeOrPerished(layer[i],*** ***Atmosphere)2***
s	***Atmosphere***
H,+,0	***Atmosphere*, ⊜,*** AtmosphereOfGas

a ⊜ b::= b

Enor(E)	***i = 1..n***
f(e)	***<Handlers[i]> if n!=i && layers[i] == layers[n]***
s	***MergeOrPerished***
H, +, 0	***LayerOfGas*, ⊕,<>***

DataToObjects():



```
DataToObjects()
for i ← 0 to gases.Count
    (gases[i].Item1)
        'Z':
            layers.Add(new Ozone(gases[i].Item2))
            break
        'X':
            layers.Add(new Oxygon(gases[i].Item2))
            break
        'C':
            layers.Add(new CarbonO2(gases[i].Item2))
            break
        default:
            "Wrong Gases Input"
for i← 0 to effect.Length
    (effect[i])
        'O':
            atmospheres.Add(Other.Instance())
            break
        'S':
            atmospheres.Add(SunShine.Instance());
            break
        'T':
            atmospheres.Add(Thunderstorm.Instance())
            break
        default:
            "Wrong AtmosphereInput"
```

A layer's part's changing into another layer means reducing the thickness of the layer and adding that remainder part after transformation to the first identical layer, or in case of there is no identical layer and the remainder thickness is greater than 0.5km creating a new layer on top, step by step.

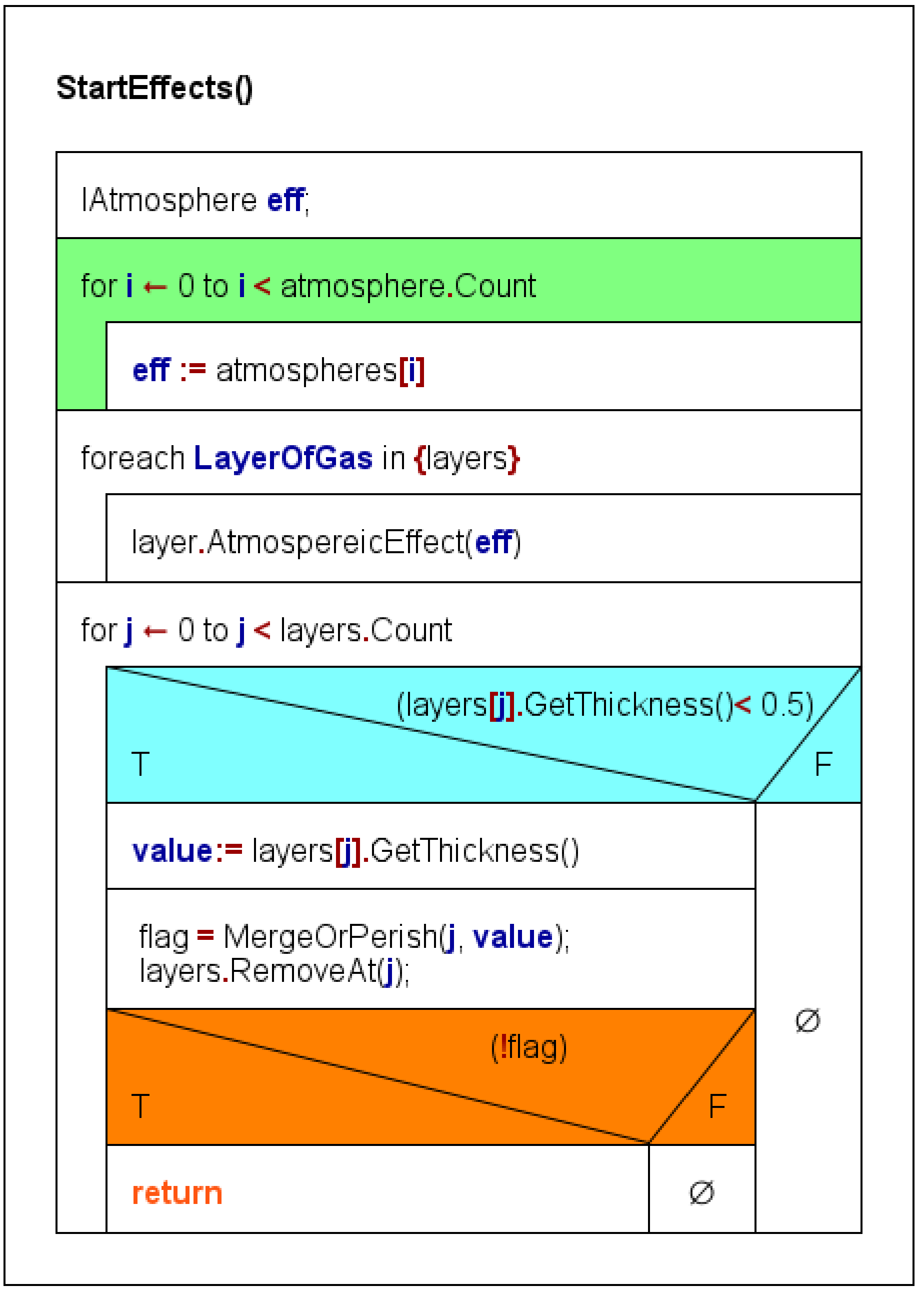
StartEffects():

The StartEffects method iterates over a list of atmospheric effects and applies each effect to all gas layers. If any gas layer's thickness falls below 0.5 after applying the effects, the method tries to merge that thin layer with another layer. If merging is not possible, the process stops.

Iterate over atmospheric effects: For each atmosphere in atmospheres, apply the effect to all gas layers.

Apply effects to gas layers: For each gas layer, call AtmospereicEffect to apply the current atmospheric effect.

Check gas layer thickness: After applying the effects, check if any gas layer's thickness is less than 0.5.

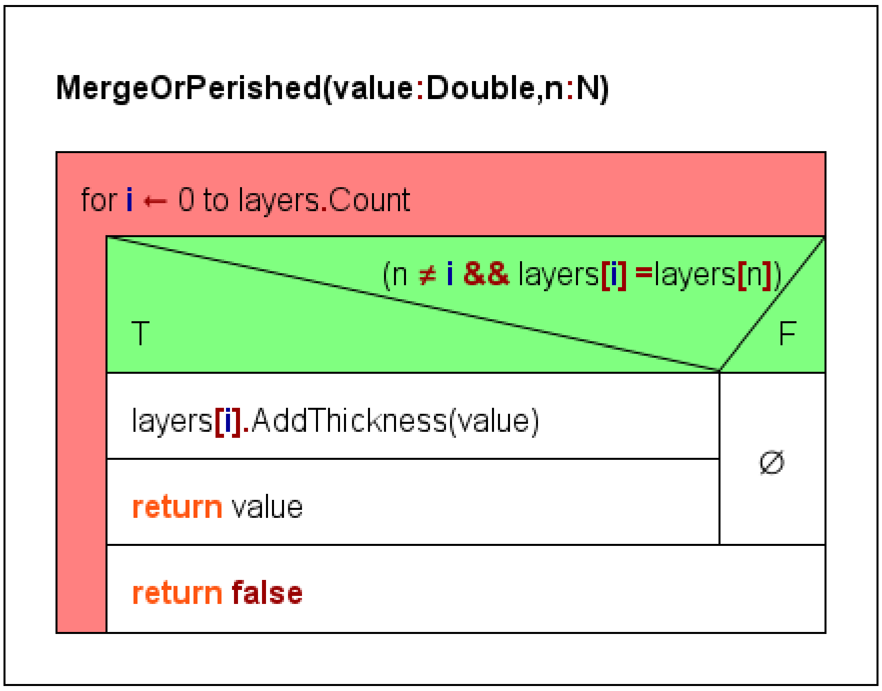


```
StartEffects()

IAtmosphere eff;
for i ← 0 to i < atmosphere.Count
    eff := atmospheres[i]
foreach LayerOfGas in {layers}
    layer.AtmospereicEffect(eff)
for j ← 0 to j < layers.Count
    (layers[j].GetThickness()< 0.5)
    T:
        value:= layers[j].GetThickness()
        flag = MergeOrPerish(j, value);
        layers.RemoveAt(j);
        (!flag)
        T:
            return
        F:
            ∅
    F:
        ∅
```

MergeOrPersihed():

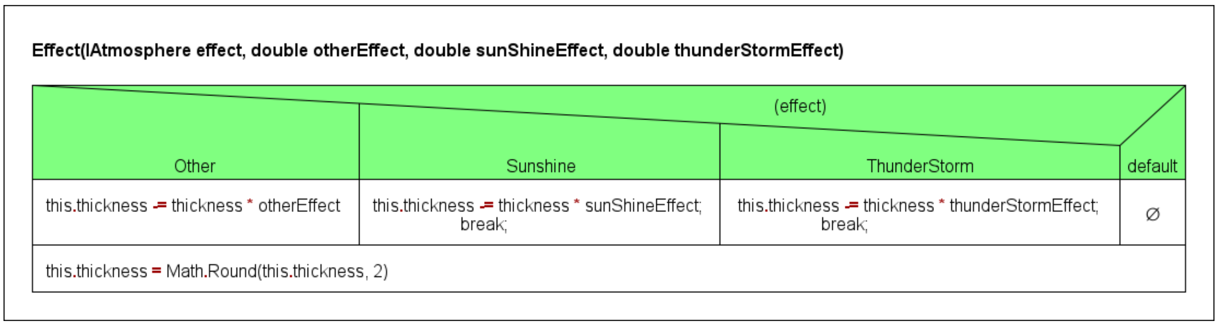
The MergeOrPerish method attempts to merge a thin gas layer (identified by its index n and thickness value) with another gas layer of the same type. It iterates through all layers, and if it finds a matching layer (one that is not the same layer at index n), it adds the thin layer's thickness to the matching layer and returns true. If no matching layer is found, it returns false.



```
MergeOrPerished(value:Double,n:N)
for i ← 0 to layers.Count
    (n ≠ i && layers[i] =layers[n])
    T:
        layers[i].AddThickness(value)
        return value
    F:
        ∅
    return false
```

Effect();

The Effect method reduces the thickness of the gas layer based on the type of atmospheric effect it receives. Depending on whether the effect is Other, SunShine, or Thunderstorm, it decreases the thickness by a specific percentage (otherEffect, sunShineEffect, or thunderStormEffect respectively). After applying the effect, it rounds the thickness to two decimal places.



```
Effect(IAtmosphere effect, double otherEffect, double sunShineEffect, double thunderStormEffect)
(effect)
    Other:
        this.thickness -= thickness * otherEffect
    Sunshine:
        this.thickness -= thickness * sunShineEffect;
        break;
    ThunderStorm:
        this.thickness -= thickness * thunderStormEffect;
        break;
    default:
        ∅
this.thickness = Math.Round(this.thickness, 2)
```

Test Cases:

Black Box Testing:

Black box testing focuses solely on the functionality of the software without considering its internal workings. Here's an example of black box testing for the AtmosphereOfgas class:

lack Box Test Case 1:

Test Name: ChangeGasLayer_EmptyList_ThrowsEmptyListException

Test Objective: Verify that an EmptyListException is thrown when attempting to change the gas layer with an empty list.

Test Input: Empty list of gases (List<Gas>())

Expected Output: EmptyListException is thrown

Black Box Test Case 2:

Test Name: ChangeGasLayer_SecondGasNotHandled_AddsNewGas

Test Objective: Verify that a new gas is added if the second gas is not handled.

Test Input: List of gases containing ozone and oxygen gases (List<Gas> { new OzoneGas(1.0), new OxygenGas(2.0) }), target gas layer "Z" for ozone gas and "C" for the new gas, multiplier of 0.5

Expected Output: New gas "C" with thickness calculated based on the multiplier is added to the list of gases

Grey Box Testing:

Grey box testing involves having partial knowledge of the internal workings of the software under test. Here's an example of grey box testing for the AtmosphereOfgas class:

Grey Box Test Case 1:

Test Name: ChangeGasLayer_HandleExistingGas_UpdatesGasThickness

Test Objective: Verify that the thickness of an existing gas is correctly updated when it is handled.

Test Input: List of gases containing ozone and oxygen gases (List<Gas> { new OzoneGas(1.0), new OxygenGas(2.0) }), target gas layer "Z" for ozone gas and "X" for oxygen gas, multiplier of 2.0

*Expected Output: Thickness of oxygen gas ("X") is updated to 4.0 (original thickness * multiplier)*

Grey Box Test Case 2:

LAYER OF GASES

Test Name: ChangeGasLayer_NotHandlingSecondGas_RemoveGasesIfThicknessLessThan0.5

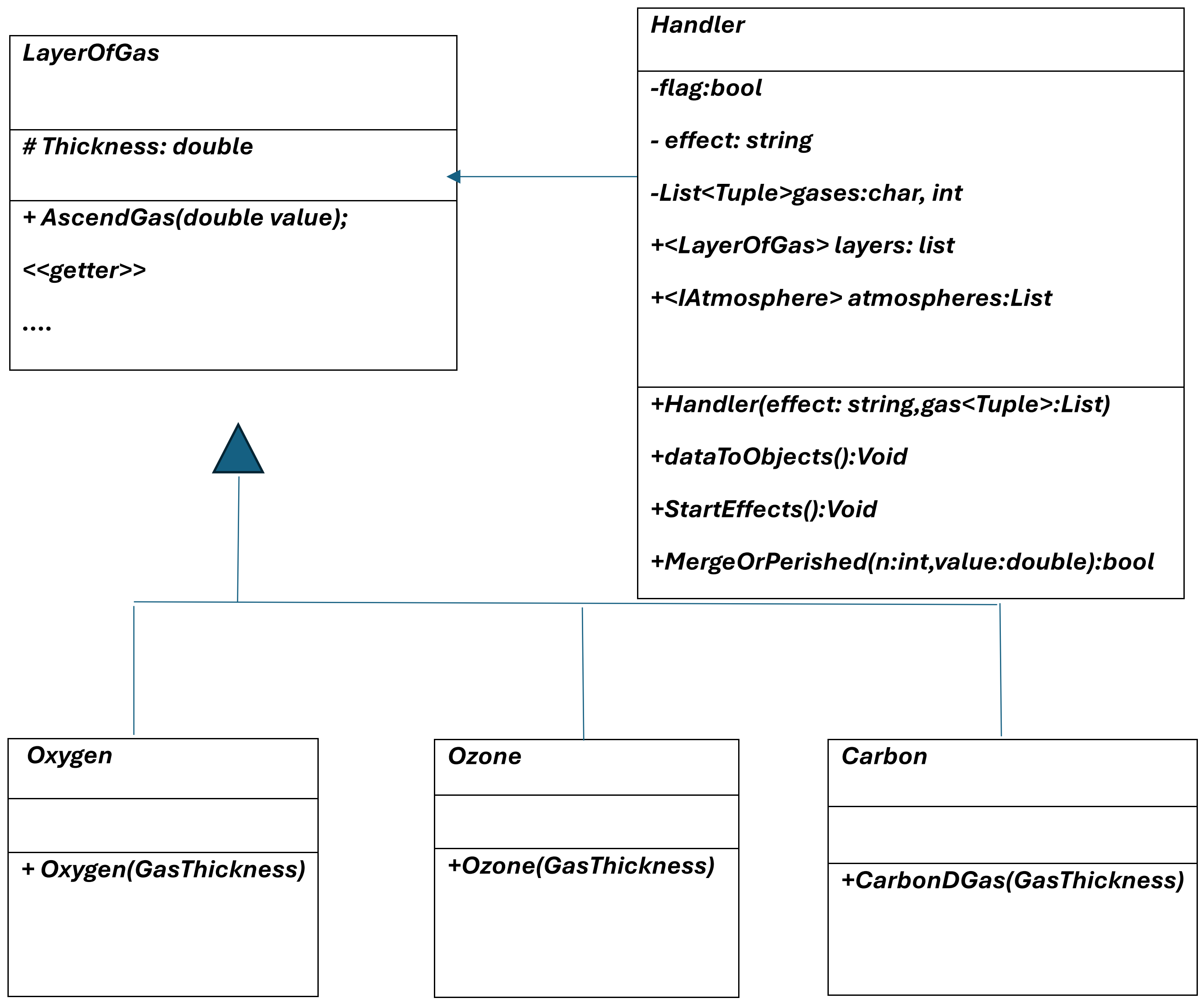
Test Objective: Verify that gases with thickness less than 0.5 are removed if the second gas is not handled.

Test Input: List of gases containing ozone and oxygen gases (List<Gas> { new OzoneGas(1.0), new OxygenGas(2.0) }), target gas layer "Z" for ozone gas and "Y" for the new gas, multiplier of 0.5

Expected Output: New gas "Y" is not added, and any gases with thickness less than 0.5 are removed from the list

LAYER OF GASES

Pattern: Factory Design Pattern



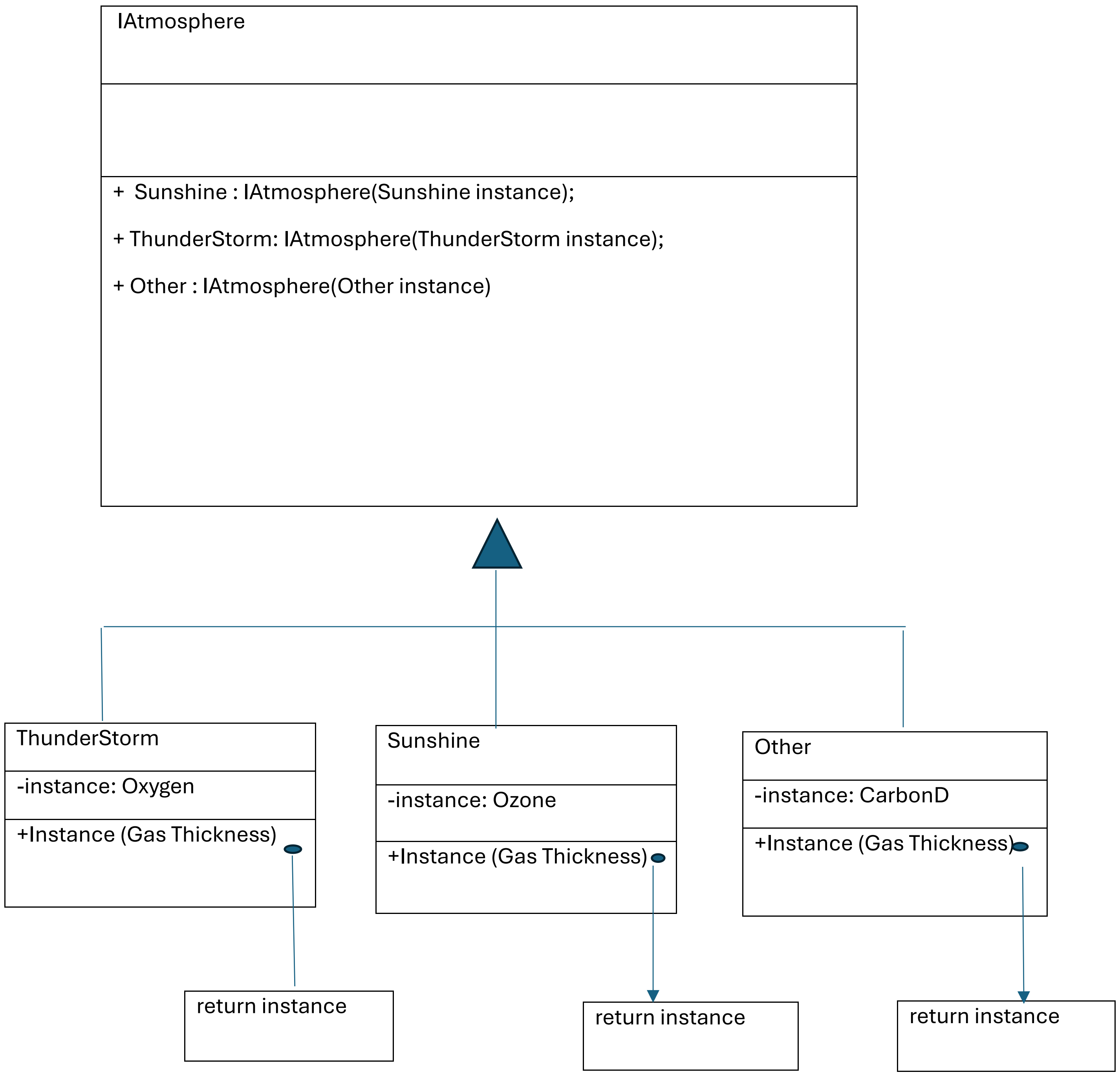
LayerOfGas
# Thickness: double
+ AscendGas(double value);
<<getter>>
....
Handler
-flag:bool
- effect: string
-List<Tuple>gases:char, int
+<LayerOfGas> layers: list
+<IAtmosphere> atmospheres:List
+Handler(effect: string,gas<Tuple>:List)
+dataToObjects():Void
+StartEffects():Void
+MergeOrPerished(n:int,value:double):bool
Oxygen
+ Oxygen(GasThickness)
Ozone
+Ozone(GasThickness)
Carbon
+CarbonDGas(GasThickness)

We can Use **Template Design Pattern** To solve this Diagram for Atmosphere Layer Variables:

In the Template Design Pattern we use the Singleton, for this We have to Implement Thunderstorm, Sunshine ,and Other to Create Instance Method for them, so we can Create Only the Atmosphere Layer Variables which we Have in the Question; i.e

1) Sunshine
2) Thunderstorm
3) Other

LAYER OF GASES


IAtmosphere
+ Sunshine : IAtmosphere(Sunshine instance);
+ ThunderStorm: IAtmosphere(ThunderStorm instance);
+ Other : IAtmosphere(Other instance)
ThunderStorm
-instance: Oxygen
+Instance (Gas Thickness)
Sunshine
-instance: Ozone
+Instance (Gas Thickness)
Other
-instance: CarbonD
+Instance (Gas Thickness)
return instance
return instance
return instance