Aryan Shams Ansari – GTPHX8

Group 5 – gtphx8@inf.elte.hu

Task:

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 at 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.

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!

Analysis:

Independent objects in the task are the different gas types. They can be divided into 3 different groups: Ozone, Oxygen, Carbon Dioxide, all of them have an ID and a Thickness field which can be changed and modified according to the atmosphere variables which are Thunderstorm, Sunshine, Other, all of them have an ID. These atmospheric variables can change the thickness of each Gas layer according to the table given below:

	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	-

Diagram:(Gas Class – types of Gas):

By using the Factory design pattern, we can always access the right Gas type when we encounter the ID of that Gas type.

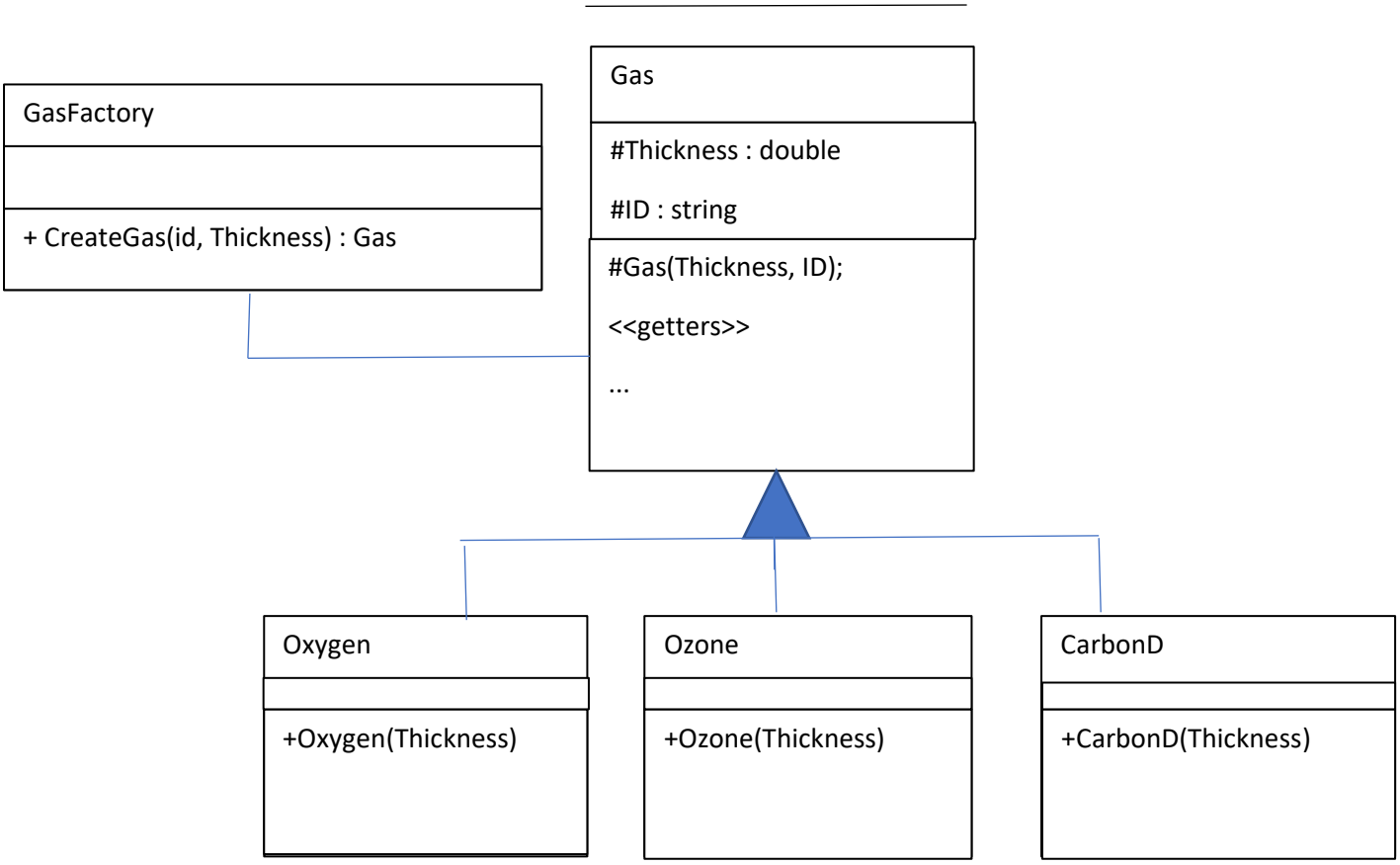
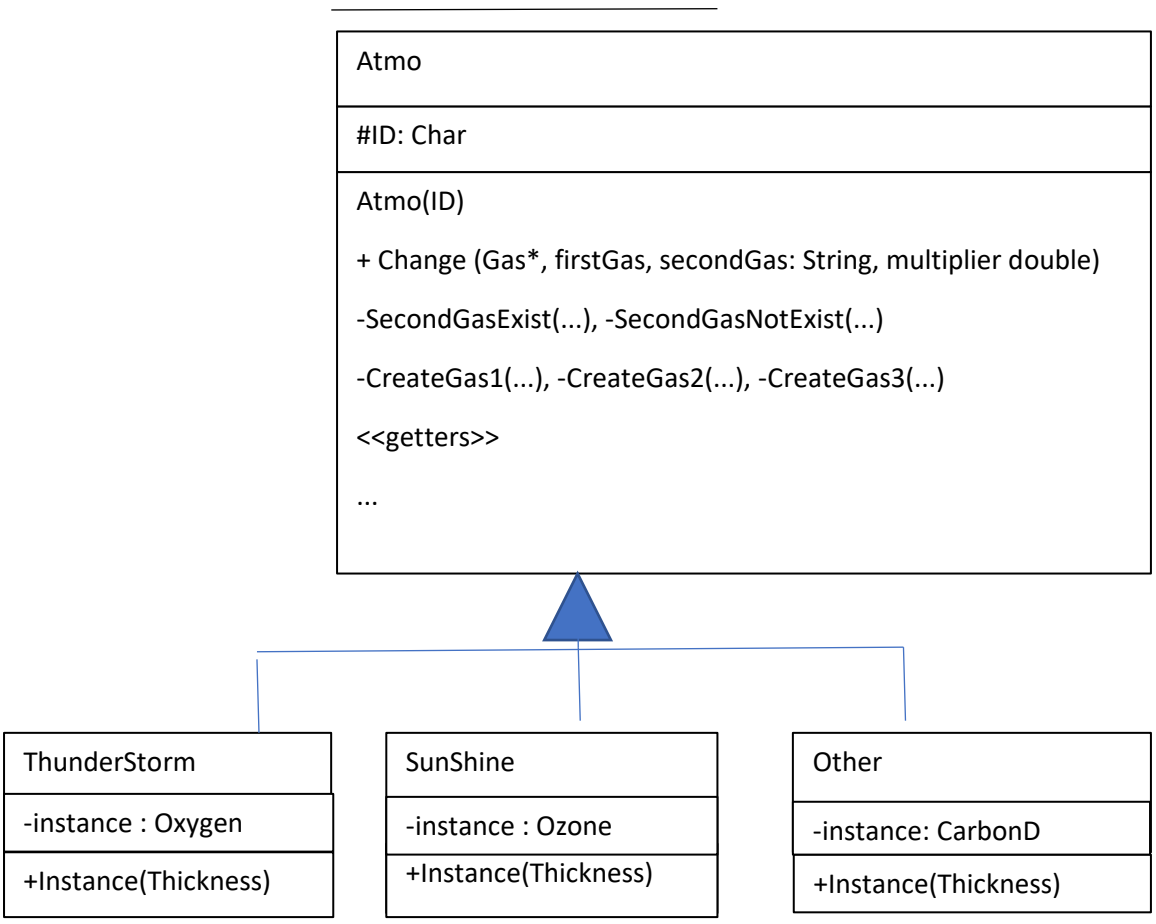


Diagram:(Atmo Class – Atmospheric variables):

By using singleton / template design patterns, we implement Thunderstorm, Sunshine, and Other and create Instance method for them, so we can create only one instance of the atmospheric variables.



Specification:

Pattern: Template Method Pattern

Short Description: The **Change** function uses the Template Method pattern to define the overall structure of the algorithm for changing gases in the list. It delegates some steps to the subclasses **`SecondGasNotExist`** and **`SecondGasExist`**.

1) function Change(gasList, firstGas, secondGas, multiplier)

```
if gasList is empty
   throw ListIsEmpty exception

for i = 0 to length of gasList - 1
   if gasList[i].getID() is equal to firstGas
      for j = i + 1 to length of gasList - 1
         if gasList[j].getID() is equal to secondGas
            SecondGasExist(gasList, firstGas, secondGas, multiplier, i, j)
         else
            SecondGasNotExist(gasList, firstGas, secondGas, multiplier, i, j)
```

2) function SecondGasNotExist(gasList, firstGas, secondGas, multiplier, i, j)

```
gasList.add(GasFactory.CreateGas(secondGas, gasList[i].getThick() * multiplier))

if gasList[last index].getThick() < 0.5
   gasList.remove(gasList[last index])

CreateGas2(gasList, multiplier, i)

if gasList[i].getThick() < 0.5
   for k = j + 1 to length of gasList - 1
      if gasList[k].getID() is not equal to firstGas
         gasList.remove(gasList[i])
      else if gasList[k].getID() is equal to firstGas
         CreateGas3(gasList, i, k)
         gasList.remove(gasList[i])
```

3) function SecondGasExist(gasList, firstGas, secondGas, multiplier, i, j)

```
CreateGas1(gasList, multiplier, i, j)
CreateGas2(gasList, multiplier, i)

if gasList[i].getThick() < 0.5
   for k = j + 1 to length of gasList - 1
      if gasList[k].getID() is not equal to firstGas
         gasList.remove(gasList[i])
      else if gasList[k].getID() is equal to firstGas
         CreateGas3(gasList, i, k)
         gasList.remove(gasList[i])
```

Pattern: Factory Method Pattern

Short Description: The **Change** function utilizes the Factory Method pattern through the **`GasFactory.CreateGas()`** method to create new **`Gas`** objects based on the provided gas names and thickness values.

function CreateGas(id: string, thickness: double) -> Gas:

```
switch id:
    case "Z":
        return create Ozone(thickness)
    case "X":
        return create Oxygen(thickness)
    case "C":
        return create CarbonD(thickness)
    default:
        throw new ArgumentException("Invalid gas type: " + id)
```

Testing:

1) By using Init() method we create two lists and we pass the following strings/chars to them.

atmoVars = { 'O', 'T', 'S' }

Layers = { "Z", "X", "C" }

2)We check if our CreateGas() method throws argument exception by giving it invalid GasTypes like "Y" and some random Thickness which results in argument exception.

3) Using the lists above we create the following test cases, and we check if in the main program we **are** encountering the same Atmospheric variables or Gas types.

CheckSunshineTEST -> Checks if in the main program method, we are returning character 'S'

CheckOtherTEST -> Checks if in the main program method, we are returning character 'O'

CheckThunderTEST -> Checks if in the main program method, we are returning character 'T'

CheckOzoneTEST -> Checks if in the main program method, we are returning string "Z"

CheckOxygenTEST -> Checks if in the main program method, we are returning string "X"

CheckCarbonDTEST -> Checks if in the main program method, we are returning string "C"

3) Using the lists above we create the following test cases, and we check if in the main program we **are not** encountering the same Atmospheric variables or Gas types.

RevCheckSunshineTEST -> Checks if in the main program method, we are returning any other character but 'S'

RevCheckOtherTEST -> Checks if in the main program method, we are returning any other character

but 'O'

RevCheckThunderTEST -> Checks if in the main program method, we are returning any other character but 'T'

RevCheckOzoneTEST -> Checks if in the main program method, we are returning any other string

but "Z"

RevCheckOxygenTEST -> Checks if in the main program method, we are returning any other string but “X”

RevCheckCarbonDTEST -> Checks if in the main program method, we are returning any other string but “C”