**Name : Aly Elalwany Neptun code : WGL13Q Group : 3 Assignment : 3**

# Task

**Teréz A. Várkonyi**

NEPTUNCODE

Group 0

**3. assignment/0. Task**

11th February 2019

*In the hydrological cycle of the Earth,various areas affect the weather as well as areas are also affected by various weathers. Areas involved in the simulation: plain, grassland, lakes region. Each area has a name, and the amount of water stored in the certain area is also given in km3. The humidity of the air over the areas is also given in percentage.*

*The possible types of weather are the following: sunny, cloudy, rainy, depending on the humidity of the air. In case the humidity exceeds 70%, the weather gets rainy and the humidity decreases to 30%.In case the humidity is between 40-70%, the calculation of the chance of rainy weather is: (humidity-40)\*3,3%, otherwise the weather is cloudy. Humidity below 40% leads to sunny weather.*

*In the following,we declare how the certain areas respond to the different type of weathers.First the amount of water stored by the area varies then the weather will be affected. There is no type of areas with negative amount of water stored.*

*In case the type is plain, if the weather is sunny, the amount of water will be decreased by 3 km3; if cloudy, it will be decreased by 1 km3; for rainy weather it will be increased by 20 km3. The humidity of the air is increased by 5%. If the amount of the stored water is greater than 15 km3, the plain area changes into grassland.In case of type grassland: in sunny weather,the amount of water is decreased by 6 km3, for cloudy it will be decreased by 2 km3, but and for rainy, it will be increased by 15 km3. The humidity of the air is increased by 10%. The area becomes lakes region obtaining amount of water over 50 km3, whereas in case the amount of stored water goes below 16 km3, the area changes to plain.In case of type lakes region: in sunny weather,the amount of water is decreased by 10 km3, for cloudy it will be decreased by 3 km3, for rainy it will be increased by 20 km3. The humidity will be increased by 15%.Beyond an amount of water of 51 km3the area changes into grassland.*

*The program reads data from a text file. The first line of the file contains asingle integer N indicating the number of areas. Each of the following N lines contains the attributes of an area separated by spaces: the owner of the area, the type of the area, and the amount of water stored by the area. In the last line, the humidity of the air is given in percentage. The type is identified by a character: P –plain, G –grassland, L –lakes region.*

***After 10 simulation rounds, determine the owner of the area which is storing the greatest amount of water. The amount of water is also required to be determined. The program should print all attributes of the certain areas by simulation rounds!***

# Analysis1

Independent objects in the task are the areas. They can be divided into 3 different groups: Plains , Grasslands and Lakes, we also have the weathers divided into 3 different groups : Sunny, Cloudy and Rainy.

All of them have a name and amount of water. We can examine what happens when the amount of water changes in correspondence to the weather. The effects on the weather and the areas are in the following way:

Sunny:

|  |  |  |
| --- | --- | --- |
| Area | amountOfWater | Humidity |
| Plain | -3 | - |
| Grassland | -6 | - |
| Lakes | -10 | - |

Cloudy:

|  |  |  |
| --- | --- | --- |
| Area | amountOfWater | Humidity |
| Plain | -1 | - |
| Grassland | -2 | - |
| Lakes | -3 | - |

Rainy:

|  |  |  |
| --- | --- | --- |
| Area | amountOfWater | Humidity |
| Plain | +20 | +5 |
| Grassland | +15 | +10 |
| Lakes | +20 | +15 |

ChangeState() table

|  |  |  |
| --- | --- | --- |
| Area | Amount of water | New state |
| Plain | >15 | Grassland |
| Grassland | <16 | Plain |
| Grassland | >50 | Lakes |
| Lakes | >51 | Grassland |

# Plan2

To describe the areas, 4 classes are introduced: base class *Area* to describe the general properties and 3 children for the concrete types: *Plain* , *Grassland*, and *Lakes*. Regardless the type of the areas, they have several common properties, like the name (*\_name*) and the amount of water (*\_amountOfWater*), the getter of its name (*name()*), the setter of water(setWater()) and the setter of name (setName()) ,if it has negative water (negativeWater*()*) and it can be examined what happens when the weather changes. This latter operation (*changeState()*) modifies the nature of the area and transmutes the current area. The method *changeState()* must be implemented just on the level of the concrete classes as its effect depends on the type of the area. Therefore, the general class *Areas* is going to be abstract, as method *changeState()* is abstract and we do not wish to instantiate such class.

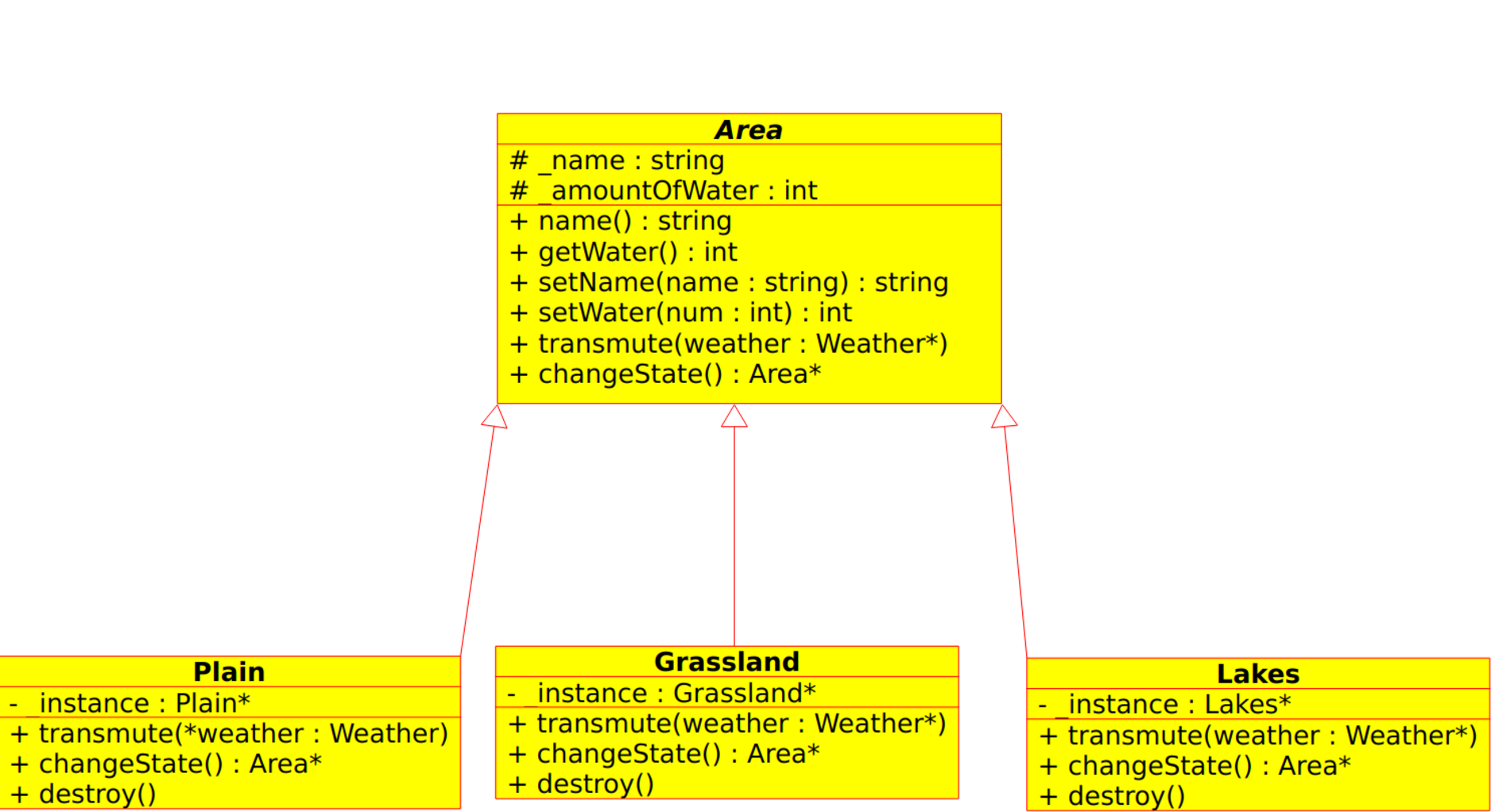
The method *transmute()* is the method that changes the amount of water in the area as well as it changes the humidity of the weather as mentioned in the Analysis section above.

General description of the areas is done the base class *Areas* from which concrete grounds are inherited: *Plain*, *Grassland*, and *Lakes*. Every concrete area has 2 methods that show how a Sunny ,a Cloudy, or a Rainy weather changes it and how the Weather may be affected by humidity as well. Objects are referred by pointers.

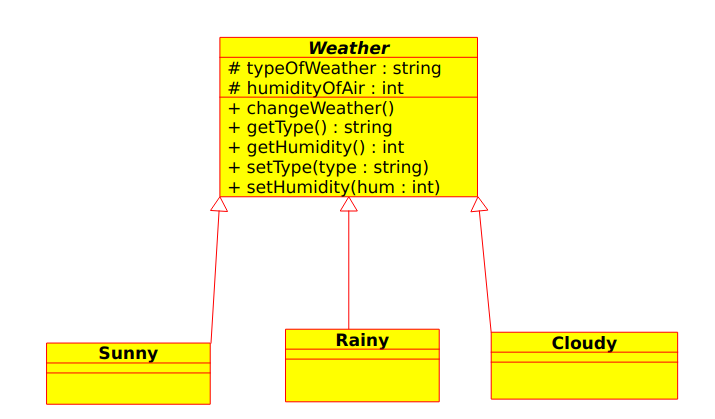
The special classes initialize the name and the amount of water through the constructor of the base class and override the operation *transmute()* in a unique way. Initialization and the override are explained in Section Analysis. According to the tables, in method *transmute()*, conditionals have to be used in which the type of the area is examined. Though, the conditionals are not effective if the program might be extended by new area types, as all of the methods *transmute()* in all of the concrete area classes have to be modified. To avoid it, design pattern Visitor is applied where the area classes are going to have the role of the visitor.

To describe the Weather, 4 classes are introduced: base class *Weather* to describe the general properties and 3 children for the concrete types: *Sunny* ,*Cloudy* , and *Rainy* . Regardless the type of the weather, they have several common properties, like the type (*typeOfWeather*) and the humidity (humidityOfAir), the getter of its type (getType*()*), the getter of humidity (getHumidity()) and the setter of type (setType()) and the setter of humidity (setHumidity())

We create the method *(changeWeather())* in order for us to change the weather according to the change in humidity based on each class. We also implemented the method *(changeWeather())* separately in each child class.

hough, the conditionals are not effective if the program might be extended by new Weather types, as all of the methods *changeWeather()* in all of the concrete weather classes have to be modified. To avoid it, design pattern Visitor is applied where the weather classes are going to have the role of the visitor.

Methods *transmute()* of an area expect a weather object as an input parameter as a visitor and calls the methods which corresponds to the species of the creature.



All the classes of the areas are realized based on the Singleton design pattern, as it is enough to create one object for each class.  
In the main file, we are doing 10 rounds of simulation using *simulation()* in this sense:

We loop through the *owners* vector that we have and then we are transmuting the area using *transmute() by* passing in a temporary pointer *temp\_ptr*  to the current weather which then will be used to change the weather to Sunny, Rainy or Cloudy.  
We are then checking if the water of the area owned by this owner is negative or not. If so, we display negative water and we return false indicating that there has been an issue with the simulation. Otherwise, we use the method *changeState()* to change the states of the current area that the owner has. The function *simulation()*  returns true upon completion.

**Analogy:**

|  |  |
| --- | --- |
| enor (E) | j=1..n |
| f(e) | changeState(owners[j].area) |
| s | owners[j].area |

A = *owners: Ownern*, *humidity:****N*** , l : **L**, temp\_ptr : Weather\*  
Pre = n>0

Post = (*for every i in 1..10 (for every j in 1..n owners[j]:=  
 owners[j].area→transmute(temp\_ptr)*

***AND***  owners[j].area→getWater() >=0 → changeState(owners[j].area) *))*

**Analogy:**

|  |  |
| --- | --- |
| enor (E) | *i*=1..10 |
| f(e) | transmute(owners[i].area, temp\_ptr) |

|  |  |  |
| --- | --- | --- |
| temp(humidity) ,temp\_ptr:=temp.weather | | |
| i=1..10 | | |
| *j=1..n* | | |
|  | *owners*[*j*].area*->* *transmute*(*temp\_ptr)* | |
| *owners[j].area→getWater()>0* | |
| *owners [j].area:= owners [j].area->changeState()* | *SKIP* |

Each owner has an area and we are going to check the changes in the amount of water this area that the owner owns. We are then going to find the owner with the max amount of water in their area.

A = *owners: Ownern*, winner:*Owner*  
Pre = n>0

Post = for every j in [1..n] (*winner,winner.amountOfWater )= MAX owners[j].area->getWater()*

Analogy : **Max search**

|  |  |  |
| --- | --- | --- |
| enor(E) | *j* = 1 .. *n* |  |
| *f*(*e*) | owner[j].area->getWater() |  |
| *s* | *winner, winner.amountOfWater* |  |
| **H**, **>** | ***N***, > |  |

|  |  |
| --- | --- |
| winner.amountOfWater : = owners[0].area→getWater(); winner.name := owners[0].name; | |
| *j* = *1 .. n* | |
|  | owners[j].area →getWater()>winner.amountOfWater  winner.amountOfWater = owners[j].area→getWater() winner = owners[j] |

# Testing

Grey box test cases:

1. length-based:
   * Empty file
   * One owner
   * Multiple owners
   * first creature survives or not the competition
   * last creature survives or not the competition

2 Negative water test

Checking winners

1 based on number of owners in the file

* + 1 owner in a 1 owner file
  + 1 owner in more than one file
  + 1 owner where 2 of same number of

1. first and last:
   * first owner is the winner
   * second owner is the winner
   * last owner is the winner