Chyong Artur
XOG5D9
xog5d9@inf.elte.hu
Group 5

1. assignment/2. Task

30th September 2023

Task

There is a planet, where different kind of plants are living. All the plants are using nutrients to live. If a plant runs out of its nutrients, it dies. Each day one radiation type can occur from the followings: alpha, delta, or no radiation. Radiations affect the plants differently based on their types. The reaction of a plant to a given radiation consists of the following: it changes its nutrient level, and affects the radiation of the next day. The radiation of the next day:

A. alpha, if the need for alpha radiation is 3 or more greater than for the delta radiation
B. delta, if the need for delta radiation is 3 or more greater than for the alpha radiation
C. no radiation, otherwise

There is no radiation on the first day…

Simulate the behaviors of the plants, and print out the radiation of the day and the properties of the plants on each day.

Properties of the plants: name (string), nutrients (integer), living (boolean). The types of the plants in the simulation: puffs, deltatree, parabush.

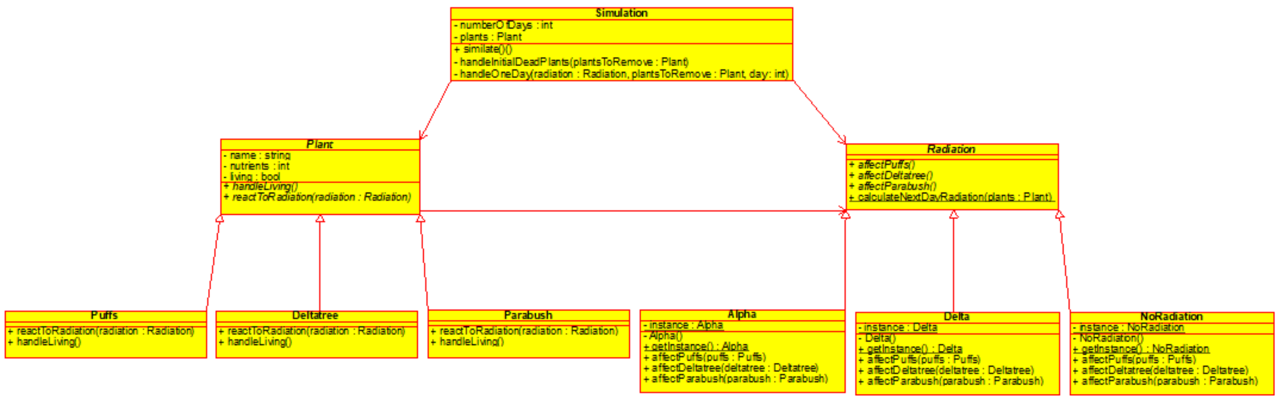
On a day of the the simulation the living plant first changes its nutrients, then if it is still alive, it can affect the radiation of the next day.

	nutrients (N)			radiation need on next day			dies
	alpha	delta	no radiation	alpha	delta	no radiation	
Puffs	+2	-2	-1	10-N			10<N
Deltatree	-3	+4	-1		+4, if N < 5 +1, if 5 ≤ N ≤ 10		
Parabush	+1	+1	-1				

Read the data of the simulation from a text file. The first line contains the number (n) of the plants. The following n lines contain the information about the plants: name, type, initial nutrient level. Type is represented by one character: p - Puffs, d - Deltratree, b - Parabush. The last line of the file defines the number of the days you have to simulate.

The program should ask for the name of the file, and it has to print out the name of the survivors (we can assume that the file is existing and its format is valid).

Class diagram


Simulation
- numberOfDays : int
- plants : Plant
+ similate()()
- handleInitialDeadPlants(plantsToRemove : Plant)
- handleOneDay(radiation : Radiation, plantsToRemove : Plant, day: int)
Plant
- name : string
- nutrients : int
- living : bool
+ handleLiving()
+ reactToRadiation(radiation : Radiation)
Radiation
+ affectPuffs()
+ affectDeltatree()
+ affectParabush()
+ calculateNextDayRadiation(plants : Plant)
Puffs
+ reactToRadiation(radiation : Radiation)
+ handleLiving()
Deltatree
+ reactToRadiation(radiation : Radiation)
+ handleLiving()
Parabush
+ reactToRadiation(radiation : Radiation)
+ handleLiving()
Alpha
- instance : Alpha
- Alpha()
+ getInstance() : Alpha
+ affectPuffs(puffs : Puffs)
+ affectDeltatree(deltatree : Deltatree)
+ affectParabush(parabush : Parabush)
Delta
- instance : Delta
- Delta()
+ getInstance() : Delta
+ affectPuffs(puffs : Puffs)
+ affectDeltatree(deltatree : Deltatree)
+ affectParabush(parabush : Parabush)
NoRadiation
- instance : NoRadiation
- NoRadiation()
+ getInstance() : NoRadiation
+ affectPuffs(puffs : Puffs)
+ affectDeltatree(deltatree : Deltatree)
+ affectParabush(parabush : Parabush)

Methods

Plant class:

The class possesses the *constructor* which accepts *String name* and *int nutrients* to set them and set *Boolean living field* to <u>*true*</u>.

For living and nutrients fields, there are *getters* and *setters* which let us modify and see the values of these fields.

Both *reactToRadiation(Radiation radiation)* and *handleLiving()* are abstract classes which will be overridden in the children classes. The first method modifies the nutrients in the accordance with the conditions given. The latter method sets living field in the accordance with the conditions given.

Puffs:

The class uses the *parent constructor*.

The method *reactToRadiation(Radiation radiation)* and *handleLiving()* are overridden.

reactToRadiation(Radiation radiation): visitor design pattern was used here. The method calls affectPuffs(this) method of the radiation class.

handleLiving(): sets living so that it is true if the current nutrients level is greater than 0 but less than or equal to 10.

Deltatree:

The class uses the *parent constructor*.

The method *reactToRadiation(Radiation radiation)* and *handleLiving()* are overridden.

reactToRadiation(Radiation radiation): visitor design pattern was used here. The method calls affectDeltatree(this) method of the radiation class.

handleLiving(): sets living so that it is true if the current nutrients level is greater than 0.

Parabush:

The class uses the *parent constructor*.

The method *reactToRadiation(Radiation radiation)* and *handleLiving()* are overridden.

reactToRadiation(Radiation radiation): visitor design pattern was used here. The method calls affectParabush(this) method of the radiation class.

handleLiving(): sets living so that it is true if the current nutrients level is greater than 0.

Radiation class:

The class possesses 3 abstract public methods with the return type of void: *affectPuffs(Puffs puffs), affectDeltatree(Deltatree deltatree), affectParabush(Parabush parabush).*

There is also a public static method with the return type of Radiation: *calculateNextDayRadiation(ArrayList<Plant> plants)*. Based on the list of plants, it calculates the radiation of the next day in the following way: if the alpha's need is 3 or more greater than delta's need then the next day will be alpha radiation day; if the delta's need is 3 or more greater than alpha's need then the next day will be delta radiation day; otherwise, it will be no radiation day.

Alpha:

The class implements the singleton design pattern. Its constructor is *private*. There is a static public method to instantiate in case the instance is null and to get the instance of this class: *getInstance()*.

affectPuffs(Puffs puffs): the method is inherited from the parent class and overridden in the following way: it adds 2 to the nutrients of the puffs.

affectDeltatree(Deltatree deltatree): the method is inherited from the parent class and overridden in the following way: it subtracts 3 from the nutrients of the deltatree.

affectParabush(Parabush parabush): the method is inherited from the parent class and overridden in the following way: it adds 1 to the nutrients of the parabush.

Detla:

The class implements the singleton design pattern. Its constructor is *private*. There is a static public method to instantiate in case the instance is null and to get the instance of this class: *getInstance()*.

affectPuffs(Puffs puffs): the method is inherited from the parent class and overridden in the following way: it subtracts 2 from the nutrients of the puffs.

affectDeltatree(Deltatree deltatree): the method is inherited from the parent class and overridden in the following way: it adds 4 to the nutrients of the deltatree.

affectParabush(Parabush parabush): the method is inherited from the parent class and overridden in the following way: it adds 1 to the nutrients of the parabush.

NoRadiation:

The class implements the singleton design pattern. Its constructor is *private*. There is a static public method to instantiate in case the instance is null and to get the instance of this class: *getInstance()*.

affectPuffs(Puffs puffs): the method is inherited from the parent class and overridden in the following way: it subtracts 1 from the nutrients of the puffs.

affectDeltatree(Deltatree deltatree): the method is inherited from the parent class and overridden in the following way: it subtracts 1 from the nutrients of the deltatree.

affectParabush(Parabush parabush): the method is inherited from the parent class and overridden in the following way: it subtracts 1 from the nutrients of the parabush.

Simulation class:

The *constructor* of the class accepts the file path. From this file, it reads the data such as number of plants, number of days for the simulation and information about the plants themselves.

The public method *simulate* simulates the environment itself by calling private methods *handleInitialDeadPlants(ArrayList<Plant> plantsToRemove)* and *handleOneDay(Radiation radiation, ArrayList<Plant> plantsToRemove, int day)* within the for loop for each day.

The private method *handleInitialDeadPlants(ArrayList<Plant> plantsToRemove)* accepts the list of plants which have to be removed from the plants because they are initially dead by conditions.

The private method *handleOneDay(Radiation radiation, ArrayList<Plant> plantsToRemove, int day)* handles one day: for every plant, it calls its method *reactToRadiation()* with the given radiation as a parameter. It calls also its method *handleLiving()*. In case the plant cannot survive, it removes this plant from the list of plants.

Testing

Test 1

Class: Plant.

Method under testing: Basic methods.

Expected behavior: Successful instantiating of plants, successful getting and setting their nutrients and living fields.

Input file: test2.txt.

Test 2

Class: Plant

Method under testing: handleLiving().

Condition: nutrients are good for surviving.

Expected behavior: set living field to true.

Input file: test2.txt.

Test 3

Class: Plant

Method under testing: handleLiving().

Condition: nutrients are not good for surviving.

Expected behavior: set living field to false.

Input file: test3.txt.

Test 4

Class: Plant.

Method under testing: reactToRadiation().

Condition: Alpha is radiation given.

Expected behavior: puffs – increase of nutrients by 2; deltatree – decrease of nutrients by 3; parabush – increase of nutrients by 1.

Input file: test2.txt.

Test 5

Class: Plant.

Method under testing: reactToRadiation().

Condition: Delta is radiation given.

Expected behavior: puffs – decrease of nutrients by 2; deltatree – increase of nutrients by 4; parabush – increase of nutrients by 1.

Input file: test2.txt.

Test 6

Class: Plant.

Method under testing: reactToRadiation().

Condition: NoRadiation is radiation given.

Expected behavior: puffs – decrease of nutrients by 1; deltatree – decrease of nutrients by 1; parabush – decrease of nutrients by 1.

Input file: test2.txt.

Test 7

Class: Radiation.

Method under testing: calculateNextDayRadiation().

Condition: Alpha's need is 3 or more greater than Delta's need.

Expected behavior: The next day is Alpha's radiation day.

Input file: test4.txt.

Test 8

Class: Radiation.

Method under testing: calculateNextDayRadiation().

Condition: Delta’s need is 3 or more greater than Alpha’s need.

Expected behavior: The next day is Delta’s radiation day.

Input file: test5.txt.

Test 9

Class: Radiation.

Method under testing: calculateNextDayRadiation().

Condition: Alpha’s need is not 3 or more greater than Delta’s need and Delta’s need is not 3 or more greater than Alpha’s need.

Expected behavior: The next day is NoRadiation’s radiation day.

Input file: test6.txt.