Munkhtenger Munkh-Aldar **2. assignment/6th. Task** 1st May 2022
WSR292
WSR292@inf.elte.hu
Group 4

Task

We are simulating the animals of the tundra. There are colonies of prey and predator animals. The number of animals in a colony affect the number of animals in other colonies. There are three predator species: the snowy owl, the arctic fox and the wolf. There are three kinds of prey: the lemming, the arctic hare and the gopher.

If the number of prey animals increase, predators can reproduce more quickly. If the number of prey is very large, most of them will wander away because they cannot find enough food. If the number of predators is large, the number of the prey decreases quicker as they are preyed upon.

Each colony has a name, a species, and the number of animals in the colony. The prey species are affected by the different predator species as follows. The number of animals in their own colony changes first, then they influence the predators.

Lemming: If they are preyed upon by a predator colony, the number of animals in their colony decreases by four times the number of animals in the predator colony. The number of animals in their colony doubles every second turn. If there are more than 200 animals in the colony, the number of animals in the colony decreases to 30.

Gopher: If they are preyed upon by a predator colony, the number of animals in their colony decreases by double the number of animals in the predator colony. The number of animals in their colony doubles every fourth turn. If there are more than 200 animals in the colony, the number of animals in the colony decreases to 40.

Hare: If they are preyed upon by a predator colony, the number of animals in their colony decreases by double the number of animals in the predator colony. The number of animals in their colony grows by 50 percent (to one and a half times their previous number) every second turn. If there are more than 100 animals in the colony, the number of animals in the colony decreases to 20.

Predators choose and attack a prey colony randomly in each turn. If there are not enough animals in the attacked colony (for example, there are not four times the number of predators in a lemming colony), the number of predators also decreases: every fourth predator out of the ones who didn't get prey perishes. Predators have offsprings every eighth turn. Normally, the snow owls have 1 offspring per 4 animals, the foxes have 3 offsprings per 4 animals, and the wolves have 2 offsprings per 4 animals.

The program should read the colonies from a text file. The first line contains the number of prey and predator colonies separated by a space. Each of the next lines contains the data of one colony separated by space: their name, their species, their starting number of animals. The species can be: o - owl, f - fox, w - wolf, l - lemming, h - hare, g - gopher.

Simulate the process until each of the prey colonies becomes extinct or the number of prey animals quadruples compared to its starting value. Print the data of each colony in each turn.

Analysis[1]

Concrete objects in the task are the prey colonies and predator colonies. Prey colonies can be divided into 3 different groups: Lemming, Gopher and Hare. Predator colonies can be divided into 3 different groups: Owl, Fox and Wolf.

All of them have a name, a species, and the number of animals in the colony that can be got. It can be examined what happens when the predator colonies attack the prey colonies each turn. As predator(s) attack randomly each turn, the prey colonies increase their number independently on certain turns and manage their number if it exceeds the limit regardless of it is getting preyed by predator or not.

Getting preyed by predator effects the prey colony in the following way:

Prey	Colony number changes
Lemming	-(4 * predator number)
Hare	-(2 * predator number)
Gopher	-(2 * predator number)

The prey number can increase on certain turns:

Prey	Every second turn	Every fourth turn
Lemming	+(lemming number)	-
Hare	+(hare number/2)	-
Gopher	-	+(gopher number)

There are different limits for each prey and their number decreases to certain numbers :

Prey	number > 200	number > 100
Lemming	number:= 30	-
Hare	-	number:= 20
Gopher	number:= 40	-

[1] This part may be skipped. It is enough to show the tables of predator changes and prey changes in the Planning section.

Predators have offsprings every eighth turn :

Predator	turn = 8
Owl	+(number / 4)
Fox	+((number / 4) * 3)
Wolf	+((number / 4) * 2)

If there are not enough animals in the attacked colony the number of predators also decreases

Attacked colony	Lemming
Predator	-((((predator number*4)-lemming number) / 4) /4)

Attacked colony	Hare
Predator	-((((predator number*2)-lemming number) / 2) /4)

Attacked colony	Gopher
Predator	-((((predator number*2)-lemming number) / 2) /4)

Plan[2]

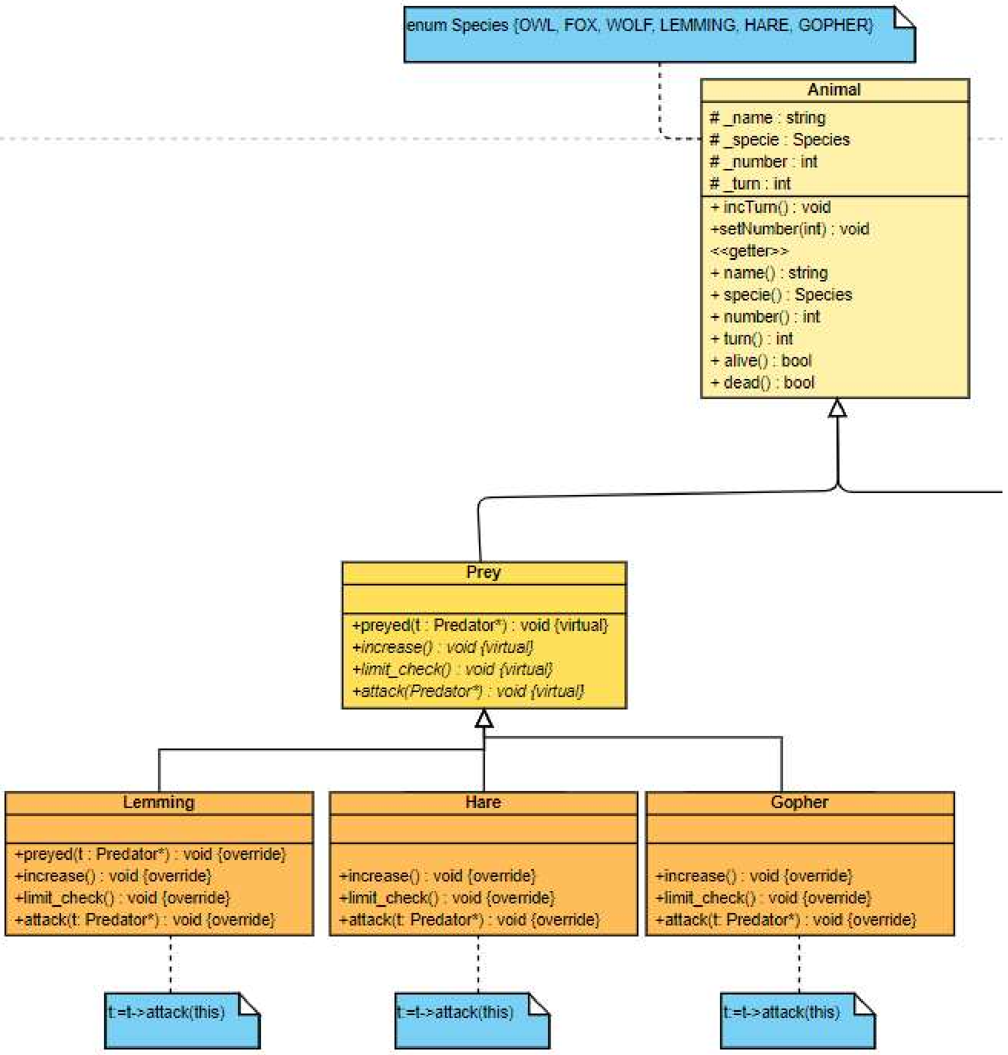
To describe the prey colonies and predator colonies, 9 classes are introduced: base class *Animal* to describe the general properties, enum which contains all possible species and 2 children for the specific colonies: *Prey* and *Predator.* 3 children which inherits *Prey* for the concrete prey species: *Lemming*, *Hare* and *Gopher.* Also, 3 children which inherits *Predator* for the concrete predator species: *Owl, Fox,* and *Wolf.* Regardless the type of the colonies, they have several common properties, like the name (*_name*), the species (*_specie*), the number (*_number*), the turn (*_turn),* the getter of its name (*name()*), the getter of its number (*number()*), the getter of its specie (*specie()*), the getter of its turn (*turn()*), if at least 1 predator and 1 prey colonies are alive (*alive()*) and it can be examined what happens when predator attacks prey colony until each prey colonies extinct or quadruples compared to starting number. Operation (*attack()*) modifies the number of the prey colony, and if prey number are not enough then predator's number decreases. Operations *alive()*, *dead() name(), specie(), number(), turn(), setNumber(), incTurn(), dead()* may be implemented in the base class already, but *attack(), preyed() increase(), limit_check()* just on the level of the concrete classes as its effect depends on species of the prey, also *offspring()* just on the level of the concrete classes as its effect depends on species of the predators. Therefore, the child classes *Prey, Predator* are going to be abstract,

[2] Plain text explanation is not necessary for the student documentations

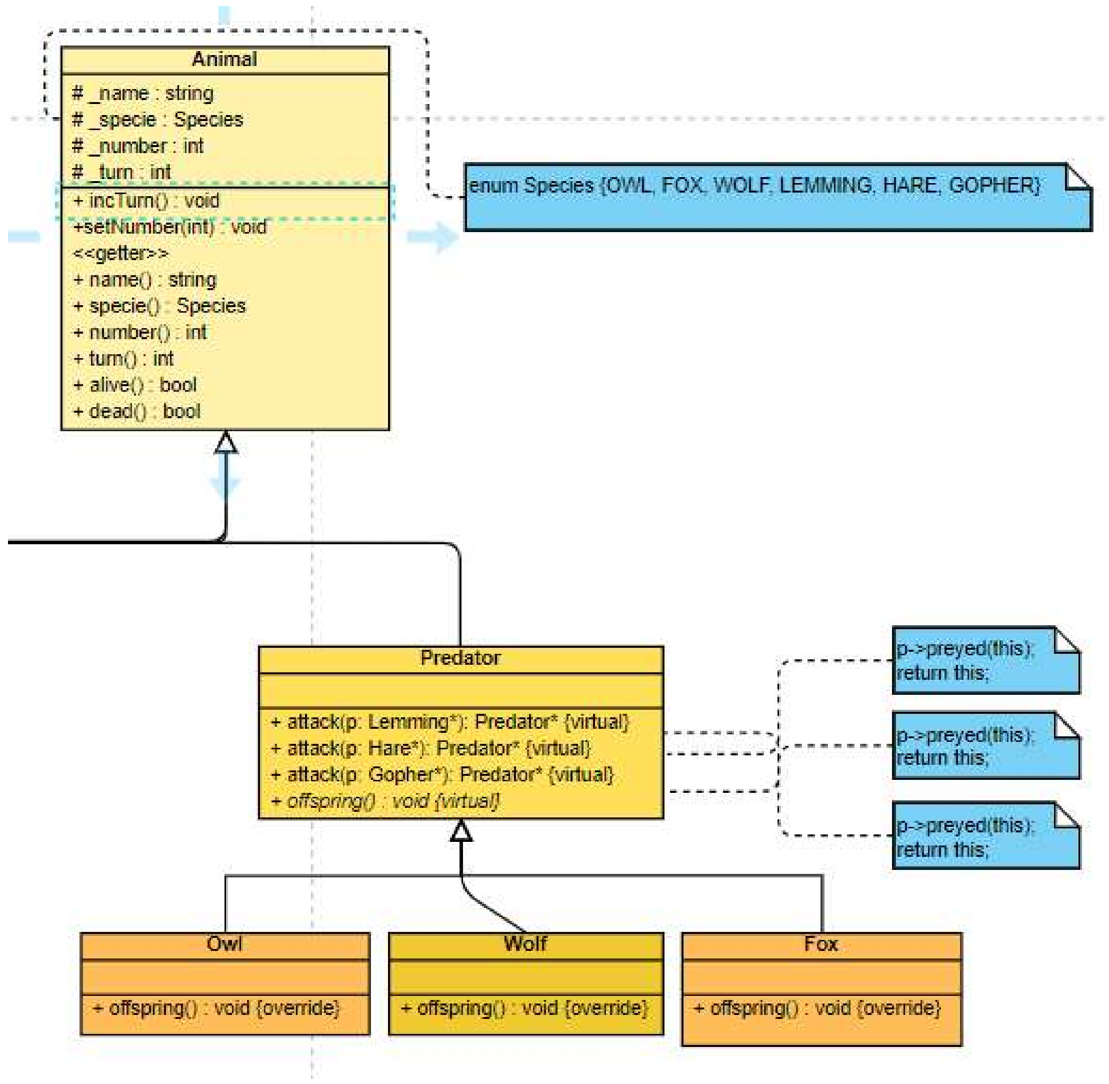
as methods *attack(), increase(), limit_check()*, *offspring()* are abstract and we do not wish to instantiate such classes.

Predator class has three methods *attack()* that depends on the concrete species of prey and changes the prey colony's number as well as show how the predator number changes when there is not enough prey. Objects are referred by pointers.

The concrete children of the prey and predator classes initialize the name, the species, the number, and the turn through the constructor of the base class Animal and override the operation *attack(), increase(), limit(), preyed() and offspring()* in a unique way. Initialization and the override are explained in Section Analysis. According to the tables, in method *attack()*, *increase(), limit(), preyed()* conditionals have to be used in which the specie of the prey is examined. In method *offspring()* conditionals have to be used in which the specie of the predator is examined. Though, the conditionals are not effective if the program might be extended by new prey specie, as all the methods in all of the concrete prey classes and predator classes have to be modified. To avoid it, design pattern Visitor is applied where the predator classes are going to have the role of the visitor.


enum Species {OWL, FOX, WOLF, LEMMING, HARE, GOPHER}
Animal
# _name : string
# _specie : Species
# _number : int
# _turn : int
+ incTurn() : void
+setNumber(int) : void
<<getter>>
+ name() : string
+ specie() : Species
+ number() : int
+ turn() : int
+ alive() : bool
+ dead() : bool
Prey
+preyed(t : Predator*) : void {virtual}
+increase() : void {virtual}
+limit_check() : void {virtual}
+attack(Predator*) : void {virtual}
Lemming
+preyed(t : Predator*) : void {override}
+increase() : void {override}
+limit_check() : void {override}
+attack(t: Predator*) : void {override}
Hare
+increase() : void {override}
+limit_check() : void {override}
+attack(t: Predator*) : void {override}
Gopher
+increase() : void {override}
+limit_check() : void {override}
+attack(t: Predator*) : void {override}
t:=t->attack(this)
t:=t->attack(this)
t:=t->attack(this)

Methods *attack()* of the concrete preys expect a Predator object as an input and calls the method which corresponds to the species of the prey. The method *preyed()* can be defined on Prey class as it is not abstract. The method can be implemented identically for Hare and Gopher classes, but Lemming class's *preyed()* method has slightly different implementation. Therefore, it is overridden in Lemming, but not overridden in Hare and Gopher which use their parent's implementation. Both *increase()* and *limit_check()* methods are implemented differently in each concrete prey classes. So, both methods are overridden in each 3 children classes.


Animal
# _name : string
# _specie : Species
# _number : int
# _turn : int
+ incTurn() : void
+setNumber(int) : void
<<getter>>
+ name() : string
+ specie() : Species
+ number() : int
+ turn() : int
+ alive() : bool
+ dead() : bool
enum Species {OWL, FOX, WOLF, LEMMING, HARE, GOPHER}
Predator
+ attack(p: Lemming*): Predator* {virtual}
+ attack(p: Hare*): Predator* {virtual}
+ attack(p: Gopher*): Predator* {virtual}
+ offspring() : void {virtual}
p->preyed(this); return this;
p->preyed(this); return this;
p->preyed(this); return this;
Owl
+ offspring() : void {override}
Wolf
+ offspring() : void {override}
Fox
+ offspring() : void {override}

As *attack()* method is not dependent on the specie of predator but dependent on the specie of prey, it is sufficient to have these methods in predator class for each prey and no need to define for Owl, Fox, Wolf for each 9 cases. However, *offspring()* method is dependent on each species of predator, so it should be overridden in the 3 children classes accordingly.

Check if each of the prey animals become extinct. Denoted by *extinctPreys*.

A = $preys : Preys^n,\ dead: \mathscr{L}$

Pre = $preys = preys_0$

Post = Pre $\wedge\ \forall i \in [1..n] : dead := \forall SEARCH_{i=1..n}$ preys[i].dead()

Get sum of each prey colony number. Denoted by *initSum*.

A = $preys : Preys^n,\ sum: \mathbb{Z}$

Pre = $preys = preys_0$

Post = Pre $\wedge\ sum := \sum_{i=1..n}$ preys[i].number()

Check if sum of number of prey animals quadruples or more compared to its starting value. Denoted by *quad*.

A = $preys : Preys^n,\ initialSum: \mathbb{Z},\ quadruple: \mathscr{L}$

Pre = $preys = preys_0\ \wedge\ initialSum = initialSum_0$

Post = Pre $\wedge\ quadruple := ((\sum_{i=1..n}$ preys[i].number() $) \geq initialSum * 4)$

Get alive prey index from the prey colonies. Denoted by *aliveInd*.

A = $preys : Preys^n,\ aliveP: int^*$

Pre = $preys = preys_0$

Post = $aliveP := \bigoplus_{\substack{i=1..n \\ \text{preys[i].alive()}}} < i >$

The predator colony attacking prey colony is denoted by function *each_turn* : Prey × Predator → Prey × Predator which gives the changed prey and predator, too.

A = $predators: Predators^m,\ preys : Preys^n,\ output: String^*$

Pre = $predators = predators_0 \wedge preys = preys_0$

Post = until (!*extinctPreys*(preys) ∧ !*quad*(preys, initSum(preys$_0$)) → (aliveInd(preys$_0$) ∧

(($\forall i \in [1..n]$: preys[i].alive() → (preys[i].incTurn() ∧ preys[i].increase())) ∧

($\forall j \in [1..m]$: predators[j].alive() → (predators[j].incTurn() ∧ predators[j].offspring() ∧

preys[aliveInd[random()]].attack(predators[j]))) ∧ ($\forall i \in [1..n]$: preys[i].alive() →

preys[i].limit_check()) ∧ output:= $\bigoplus_{i=1..n}$ < prey[i] > ∧ output:= $\bigoplus_{j=1..m}$ < predators[j] >)

Analogy

extinctPreys function

enor(*E*)	*i = 1 .. n*
f (*e*)	preys[i].dead()
l	*dead*

initSum function

enor(E)	*i* = 1 .. *n*
f (*e*)	preys[i].number()
s	*sum*
H, +, 0	ℤ, +, 0

quad function

enor(E)	*i* = 1 .. *n*
sum	preys[i]
l	sum ≥ *initialSum* * 4

aliveInd function

enor(*E*)	*i = 1 .. n*
f(*e*)	i
s	*aliveP*
H, +, 0	int^*, ⊕, <>

output

enor(*E*)	*i = 1 .. n*
f(*e*)	preys[i]
s	output
H, +, 0	$animal^*$, ⊕, <>

output

enor(*E*)	*j = 1 .. m*
f(*e*)	predators[j]
s	output
H, +, 0	*animals**, $\oplus$, <>

```
output := <>
initialSum:=initSum(preys)
(!extinctPreys(preys) ∧ !quad(preys, initialSum))
    aliveIndices:=aliveInd(preys)
    i:= 1..n
        preys[i].alive()
            preys[i].incTurn()
            preys[i].increase()
    j:= 1..m
        predators[j].alive()
            predators[j].incTurn()
            predators[j].offspring()
            preys[aliveIndices[random()]].attack(predators[j])
    i:= 1..n
        preys[i].alive()
            preys[i].limit_check()
    i:= 1..n
        output:= output ⊕ preys[i]
    j:= 1..m
        output:= output ⊕ predators[j]
```

As prey colonies are independent objects, until all preys are not dead or their colony number got quadrupled compared to the starting value, prey will continue to increase their number and limit their number as long as the colony is alive. When predator randomly chooses the prey colony, first it will increment its turn by 1 and check if it is time for offsprings, nextly it will attack the prey and decrease their number. If attacked prey colony does not have enough number to get eaten by predator, prey colony will die, and predator colony number will decrease by every fourth one which it didn't get prey. Even after being preyed, if the prey colony still has more number than their limits, their number will set to certain numbers. After that, regardless of whether being alive or not, the prey colonies and predator colonies' data would be concatenated to the output. As shown above, the simulation would go until each of the prey colonies becomes extinct or the number of prey animals quadruples compared to its starting value.

Method preyed() is defined in prey class. Moreover, it is a virtual method, so it can be overridden in the child class.

```
prey.number() < predator.number()*2
    predator.number() ≤ (((predator.number() * 2 – prey.number()) / 2) / 4)
        prey.setNumber(0)
    else
        predator.setNumber(predator.number()- ((( predator.number() * 2 - prey.number()) / 2) / 4))
    prey.setNumber(0);
else
    prey.setNumber( prey.number() - (predator.number() * 2))
```

Method preyed() overridden in Lemming class.

```
prey.number() < predator.number()*4
    predator.number() ≤ (((predator.number() * 4 – prey.number()) / 4) / 4)
        prey.setNumber(0)
    else
        predator.setNumber(predator.number()- (((predator.number() * 4- prey.number()) / 4) / 4))
    prey.setNumber(0);
else
    prey.setNumber( prey.number() - (predator.number() * 4))
```

Testing

Grey box test cases:

Outer loop:

1. Length based:
   - One colony of both prey and predator
   - Many colonies of both prey and predator
2. One prey colony:
   - Zero predator colony
   - One predator colony
   - Many predators colonies

Inner loop (conditional cases):

1. Every fourth predator perishes when there is not enough prey (1 prey colony, 1 predator colony) :
   - 25 Owl, 1 Lemming
   - 25 Fox, 1 Lemming
   - 25 Wolf, 1 Lemming

2. Check if the prey number is initially 0 so it gives no output (1 prey colony, 1 predator colony) :
   - 100 Wolf, 0 Lemming
   - 100 Owl, 0 Lemming
   - 100 Fox, 0 Lemming

3. Check if the predator number is 0 so it stops when prey's number quadruples or more (1 prey colony, 1 predator colony) :
   - 0 Wolf, 2 Lemming
   - 0 Wolf, 2 Hare
   - 0 Wolf, 2 Gopher

All cases with certain outcomes (1 predator colony and 1 prey colony):

1. Only 1 Lemming colony:
   - Only 1 Wolf colony
   - Only 1 Owl colony
   - Only 1 Fox colony

2. Only 1 Hare colony:
   - Only 1 Wolf colony
   - Only 1 Owl colony
   - Only 1 Fox colony

3. Only 1 Gopher colony:
   - Only 1 Wolf colony
   - Only 1 Owl colony
   - Only 1 Fox colony