OOP Documentation Assignment 2.
Khurova Madina
Terez A. Varkonyi 2.assignment/9.task. 8th May 2023
D6XNZX
madi.hurova04022003@gmail.com
Group 13

Task

Hobby animals need several things to preserve their exhilaration. Cathy has some hobby animals: fishes, birds,and dogs. Every animal has a name and their exhilaration level is between 0 and 100 (0 means that the animals die). If their keeper is in a good mood, she takes care of everything to cheer up her animals, and their exhilaration level increases: of the fishes by 1, of the birds by 2, and of the dogs by 3. On an ordinary day, Cathy takes care of only the dogs (their exhilaration level does not change), so the exhilaration level of the rest decreases: of the fishes by 3, of the birds by 1. On a bad day, every animal becomes a bit sadder and their exhilaration level decreases: of the fishes by 5, of the birds by 3, and of the dogs by 10. Cathy's mood improves by one if the exhilaration level of every alive animal is at least 5. Every data is stored in a text file. The first line contains the number of animals. Each of the following lines contain the data of one animal: one character for the type (F – Fish, B – Bird, D – Dog), name of the animal (one word), and the initial level of exhilaration.

In the last line, the daily moods of Cathy are enumerated by a list of characters (g – good, o – ordinary, b – bad). The file is assumed to be correct. Name the animal of the lowest level of exhilaration which is still alive at the end of the simulation. If there are more, name all of them!

Analysis

Independent objects in the task are the animals. They can be divided into 3 different groups: Fishes, Birds, and Dogs. All of them have a name and an exhilaration power that can be got. It can be examined what happens to the exhilaration level of the animals when the owner (in this case, Cathy) is in a good, bad, or in an ordinary mood. The owner's mood effects the animal and its exhilaration level in the following way:

Fish

Mood	Exhilaration
GOOD	1
ORDINARY	-3
BAD	-5

Dog

Mood	Exhilaration
GOOD	3
ORDINARY	0
BAD	-10

Bird

Mood	Exhilaration
GOOD	2
ORDINARY	-1
BAD	-3

Plan

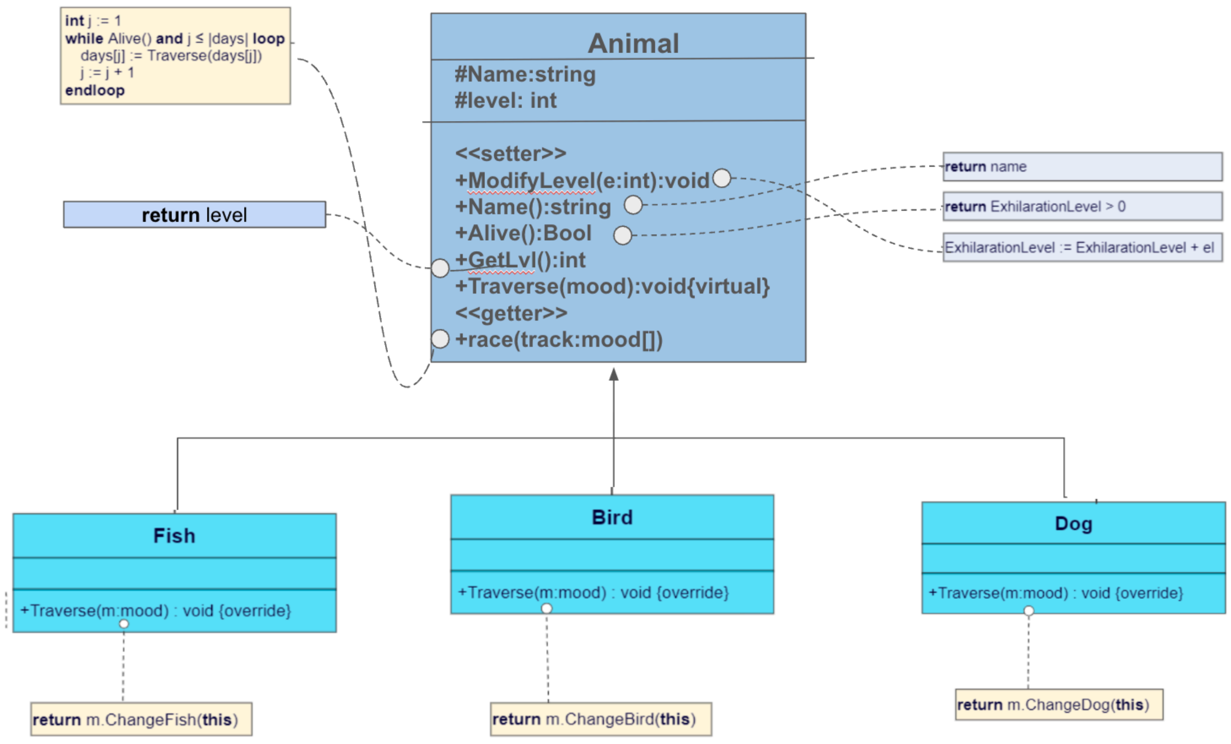
To describe the animals, 4 classes are introduced: base class Animal to describe the general properties and 3 children for the concrete types of Animals: Fish, Bird, and Dog. Regardless of the type of the Animals, they have several common properties, like the name(_name) and the power (_power), the getter of its name (name()), if it is alive (Alive()) and it can be examined what happens to the animal's exhilaration level when the Cathy's mood changes. This latter operation (Traverse()) modifies the Exhilaration Level of the Animal according to the Cathy's mood.

Operations alive() and name() may be implemented in the base class already, but Traverse() just on the level of the concrete classes as its effect on the animal's Exhilaration Level depends on the mood of the Cathy. Therefore, the general class Animal is going to be abstract, as method Traverse() is abstract and we do not wish to instantiate such class.

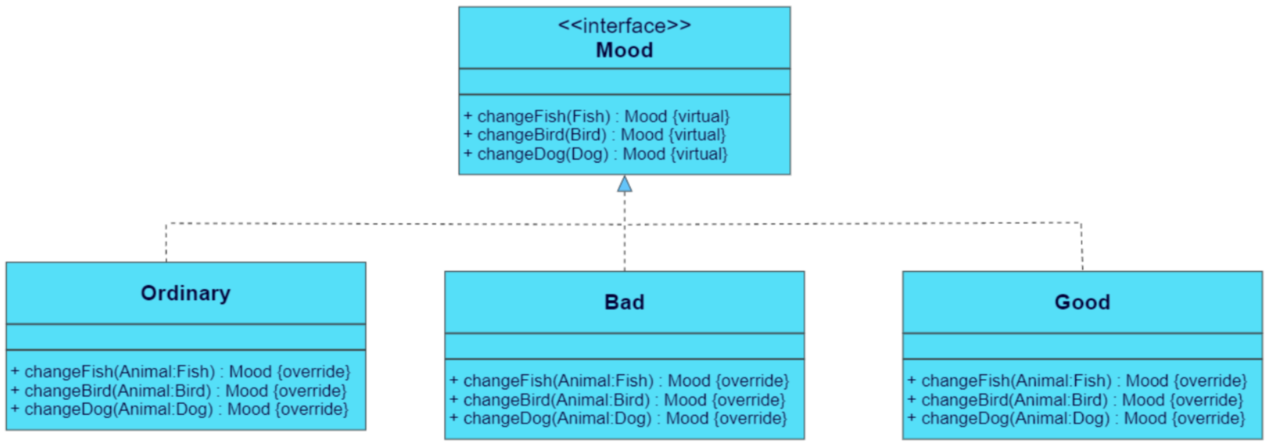
General description of each of the mood is done in the base class Mood from which the concrete moods are inherited: Fish, Bird, and Dog. Every concrete mood has three methods that show how the Exhilaration Level of a Fish, a Bird, or a Dog changes during each of the Cathy's mood, namely Ordinary, Good and Bad.

The special Animal classes initialize the name and the Exhilaration Level through the constructor of the base class and override the operation Traverse() in a unique way.

Initialization and the override are explained in Section Analysis. According to the tables, in method Traverse(), conditionals could be used in which the type of the mood would be examined. Though, the conditionals would violate the SOLID principle of object-oriented programming and are not effective if the program might be extended by new mood types, as all of the methods Traverse() in all of the concrete creature classes should be modified. To avoid it, the Visitor design pattern is applied where the mood classes are going to have the role of the visitor.


int j := 1
while Alive() and j ≤ |days| loop
days[j] := Traverse(days[j])
j := j + 1
endloop
return level
Animal
#Name:string
#level: int
<<setter>>
+ModifyLevel(e:int):void
+Name():string
+Alive():Bool
+GetLvl():int
+Traverse(mood):void{virtual}
<<getter>>
+race(track:mood[])
return name
return ExhilarationLevel > 0
ExhilarationLevel := ExhilarationLevel + el
Fish
+Traverse(m:mood) : void {override}
Bird
+Traverse(m:mood) : void {override}
Dog
+Traverse(m:mood) : void {override}
return m.ChangeFish(this)
return m.ChangeBird(this)
return m.ChangeDog(this)

Methods Traverse() of the concrete Animals expect a mood object as an input parameter as a visitor and call the methods which corresponds to the species of the Animal.


<<interface>>
Mood
+ changeFish(Fish) : Mood {virtual}
+ changeBird(Bird) : Mood {virtual}
+ changeDog(Dog) : Mood {virtual}
Ordinary
+ changeFish(Animal:Fish) : Mood {override}
+ changeBird(Animal:Bird) : Mood {override}
+ changeDog(Animal:Dog) : Mood {override}
Bad
+ changeFish(Animal:Fish) : Mood {override}
+ changeBird(Animal:Bird) : Mood {override}
+ changeDog(Animal:Dog) : Mood {override}
Good
+ changeFish(Animal:Fish) : Mood {override}
+ changeBird(Animal:Bird) : Mood {override}
+ changeDog(Animal:Dog) : Mood {override}

All the classes of the moods are realized based on the Singleton design pattern, as it is enough to create one object for each class.

Specification

In the specification, it is necessary to calculate with the n+1 versions of the track as every animal changes it. The 0th version is the initial track. The crossing of one creature is denoted by function traverse : Animal× Mood^m → Animal × Mood^m which gives the changed creature and ground, too. i th version of the track is denoted by track_i, which the program is not going to show, it is going to be just a temporal value of variable track.

A = (track:Mood^m, animals: Animal^n, alive:String*

Pre= animals = animals' and track = track'

Post= track = track_n ∧ $\forall i \in [1..n]$: animals[i], $track_i$ = race(animals[i], $track_{i-1}$) ∧

$min = MIN_{i=1..n} < animals[i].\text{Level} >$

animals[I].alive()

Analogy

first component of the value of function race()

enor(E)	i=1..n
f(e)	race(animals[I],track)
s	animals
H,+,0	Animal*, ⊜,animals[I]

second component of the value of function race()
a ⊜ b ::= b

enor(E)	i=1..n
f(e)	race(animals[I],track)
s	track
H,+,0	Mood*, ⊜,track

Minimal search analogy

enor(E)	i=1..n
f(e)	animals[I].alive()
s	min
H,<	Animal*, <

By merging the above to the same loop, the solution is got:
Note: instead of a Animal, only its name is kept.

```
min := animals[1].Level
minName := animals[1].Name
i=2..n
    animals[i].alive Λ min > animals[i].Level
        min = animals[i].Level
        minName := animals[i].Name
    else
        SKIP
```

The i^{th} animal is going to have m+1 states as moods change which is, in essence, rebuilt (from $moods_{i-1}$ to $moods_i$). 0^{th} state of animals[i] is the given animal ($animals_0[i]$), while the m^{th} state is the animal after crossing the whole moodSequence ($animals_m[i]$ = animals[i]). The i^{th} animal before Cathy has the j^{th} mood is denoted by $animal_{j-1}[i]$ from which the j^{th} state is created by method moodSequence() ($animals_j[i]$) along with the i^{th} state of the moods $moods_i[j]$. So, the task to be solved is:

"jÎ[1..m]: $animals_j[i]$, $moods_i[j]$ = moodSequence ($animals_{j-1}[i]$, $moods_{i-1}[j]$) ∧ animals[i] = $animals_m[i]$

A animal's experiencing mood means changing the animal's exhilaration level step by step, while the concatenation of the traversed moods is done, too. Both of them are based on Summation with the same enumerator (j=1 .. m):

enor(E)	***j = 1 .. m***
f(e)	***traverse(animals[I],track[j])$_1$***
s	***animals[i]***
H, +, 0	***Animals*, ⊜, animals[i]***

first component of the value of function traverse*()*
a ⊜ b ::= b

enor(E)	***j = 1 .. m***
f(e)	***traverse(animals[i], track[j])$_2$***
s	***track[j]***
H, +, 0	***Mood*, ⊕, <>***

second component of the value of function traverse()

By merging them into the same loop:

```
animals[i], track:= race(animals[i], track)
j = 1 .. m
    animals[i], track[j]) := traverse(animals[i], track[j])
```

If it is recognized that neither the animal, nor the mood are change after the death of the animal,

the effectiveness of the above algorithm may be improved:

animals[i], moods:= moodTraverse (animals[i], moods)

```
j := 1
animals[i].alive() ∧ j ≤ lowest
    animals[i], track[j] := traverse(animals[i], track[j])
    j := j+1
```

Testing

1.Assigning: Tests the correct assignment of values to an instance of the Fish class. It verifies that the name and level of the created Fish object match the expected values.

2.ModifyingLevel: Tests the modification of the level of a Fish object using the Good mood. It verifies that the level of the Fish object increases by one after applying the mood.

3.ModifyingList: Tests the modification of the levels of multiple Bird objects in a list using the Bad mood. It applies the mood to each bird in the list and compares the resulting levels with the expected levels.

4.TestMinMethod: Tests the behavior of adding a Dog object to a list. It verifies that the count of the list increases by one after adding the Dog object.