Name: Mamatova Bermet
Date: May 6th, 2024
Neptune code: BTA5BP
Group:6
Email: bta5bp@inf.elte.hu

Assignment 2/Task 9

Problem 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 dies).

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 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: fish, birds, and dogs.
All of them have a name and an exhilaration level. It can be examined what happens when the owner has different moods each day. Every mood affects each animal in the following way:
Fish:

Mood	Exhilaration level
Good	+1
Ordinary	-3
Bad	-5

Bird:

Mood	Exhilaration level
Good	+2
Ordinary	-1
Bad	-3

Dog:

Mood	Exhilaration level
Good	+3
Ordinary	*does not change
Bad	-10

Plan

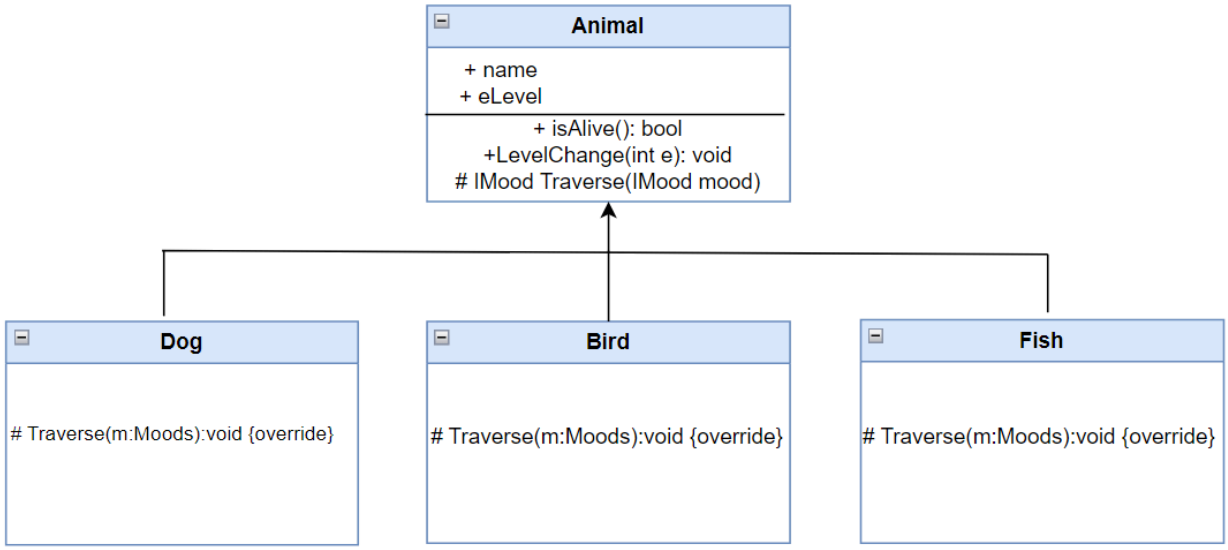
In order to characterize animals, we introduce four classes: the base class Animal encapsulates general attributes, and three subclasses Fish, Bird, and Dog represent specific types of animals. All animals, regardless of their type, share common properties such as name and eLevel (exhilaration level). They also have methods to get their name (name()) and to check if they are alive (isAlive()). Another important method is LevelChange(), which adjusts the animal's eLevel based on the mood of the carekeeper.

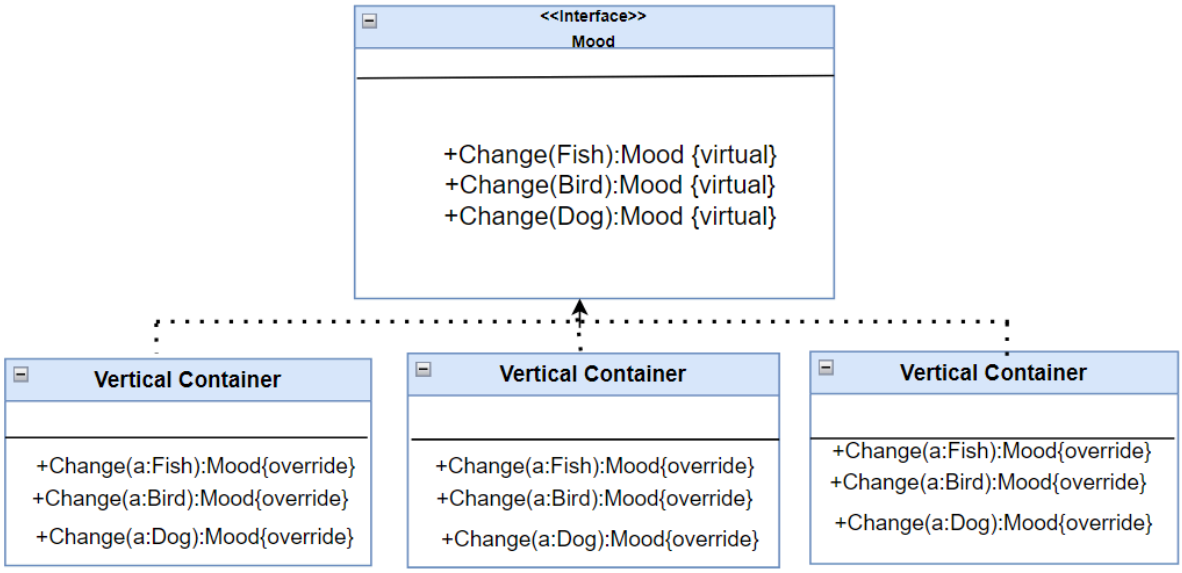
The methods alive() and name() can be implemented in the Animal base class, but LevelChange() needs to be implemented in each subclass because its effect on eLevel depends on the carekeeper's mood. As a result, Animal is an abstract class because it contains the abstract method LevelChange() and we don't want to create instances of it.

The base class Mood provides a general description of each mood, and specific moods are inherited from it: Fish, Bird, and Dog. Each specific mood class has three methods that demonstrate how the eLevel of a Fish, Bird, or Dog changes with each carekeeper's mood: Ordinary, Good, and Bad.

The specialized Animal classes initialize name and eLevel using the constructor of the base class and uniquely override the Change() operation. The initialization and override processes are detailed in the Analysis section.

In the LChange() method, conditionals could be used to check the type of mood. However, this would violate the SOLID principles of object-oriented programming and would not be efficient if new mood types were added, as all LevelChange() methods in all specific animal classes would need to be modified. To circumvent this, the Visitor design pattern is used, with the mood classes acting as the visitor.


Animal
+ name
+ eLevel
+ isAlive(): bool
+LevelChange(int e): void
IMood Traverse(IMood mood)
Dog
Traverse(m:Moods):void {override}
Bird
Traverse(m:Moods):void {override}
Fish
Traverse(m:Moods):void {override}


<<Interface>>
Mood
+Change(Fish):Mood {virtual}
+Change(Bird):Mood {virtual}
+Change(Dog):Mood {virtual}
Vertical Container
+Change(a:Fish):Mood{override}
+Change(a:Bird):Mood{override}
+Change(a:Dog):Mood{override}
Vertical Container
+Change(a:Fish):Mood{override}
+Change(a:Bird):Mood{override}
+Change(a:Dog):Mood{override}
Vertical Container
+Change(a:Fish):Mood{override}
+Change(a:Bird):Mood{override}
+Change(a:Dog):Mood{override}

Specification:

$$A = (moods : mood^{days}, animals : Animal^{n}, alive : string^{*})$$

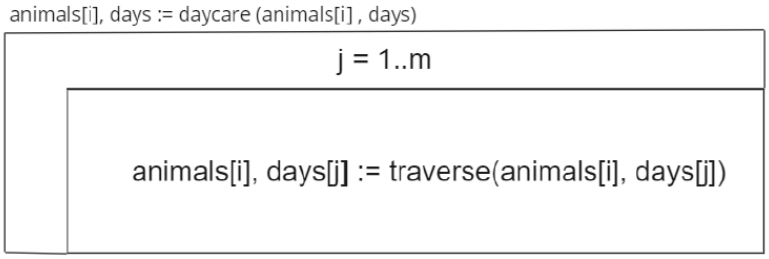
$$Pre = (animals = animals_0 \wedge moods = moods_0)$$

$$Post = moods = moods_n \wedge \forall i \in [1..n] : animals[i], moods_i = daycare(animals[i], moods_{i-1}) \quad \wedge$$

$$minexlevel = \underset{\text{animals[i].ExhilarationLevel}}{\text{MIN}_{i=1..n}} < animals[i].name >$$

When Cathy's mood changes with either ordinary,good or bad, the exhilaration level of each animal is changed. So the second task to be solved is below:

$$\forall j \in [1..m] : animals_j[i], days_i[j] = traverse(animals_{j-1}[i], days_{i-1}[j]) \wedge animals[i] = animals_m[i]$$



Testing

- NoAnimal(): This test checks if the program correctly handles a scenario where there are no animals. It asserts that the count of animals should be 0.
- OneAnimal(): This test verifies the program's behavior when there is one animal, specifically a dog named "Lassie". It asserts that the name of the animal is "Lassie" and that there is only one animal in the list.
- MoreThanOneAnimal(): This test checks the program's functionality when there are more than one animals. It asserts that there are two animals, both are dogs, and their names are "Lassie" and "Akdosh". It also checks that both animals are alive.
- AllDead(): This test is designed to verify the program's behavior when all animals are dead. It asserts that the count of alive animals should be 0.

Each test method creates a list of Animal objects, calls the Start method of the oop2.Program class with a specific input file, and then performs assertions based on the expected outcomes. The input files ("input1.txt", "input.txt", "input2.txt", "input3.txt") presumably contain data that simulate different scenarios for the tests. The Assert class from the Microsoft.VisualStudio.TestPlatform.TestHost namespace is used for the assertions.