	**Carátula para entrega de prácticas**
Facultad de Ingeniería	Ingeniería en Computación

Profesor(a): _____

Asignatura: Programación Orientada a Objetos

Grupo: 1

No de Práctica(s): 9

Integrante(s): 11700202-9

_____

_____

_____

No. de Equipo de cómputo empleado: _____

Semestre: 2024-1

Fecha de entrega: 29 de octubre de 2023

Observaciones: _____

_____

CALIFICACIÓN: _____

Index.

Practice 9 - UML Modeling.

UML diagrams of the seventh and eighth practices.

1.- Introduction.

This report has the objective of explaining the ninth practice of the object oriented programming subject, in which we must explore UML modeling for object oriented systems.

We must create four diagrams for each practice: Class diagram, object diagram, sequence diagram and states diagram.

1.1.- Hypothesis

The diagrams selected should describe the entire functioning of our programs, as a system and in runtime. It may be really tedious as we need to make a lot of sub diagrams in each of our practices.

2.- Development

2.1.- Class diagrams

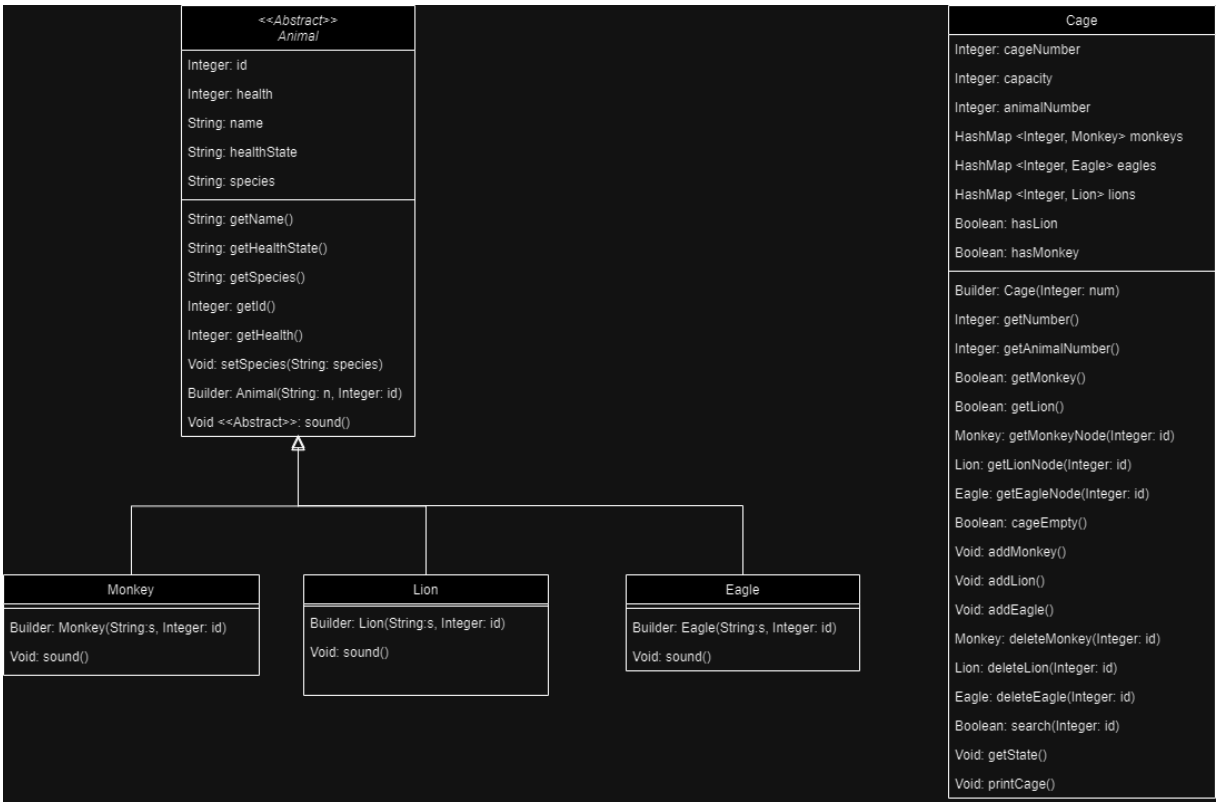

Figure 1: Class diagram for "ZooManagerByCheetos"

We have an abstract class for better inheritance handling, as we don't need to declare instances of a pure animal. We need to specify that it is abstract before the name. Then we declare all father class attributes and methods (builder included), also specifying the abstract status for the sound method (will be overridden).

Then we inherit the classes. We need to only specify the new attributes and methods (also the polymorphic methods).

The isolated Cage class has some object attributes such as HashMap, so we need to declare them with their respective keys and values.

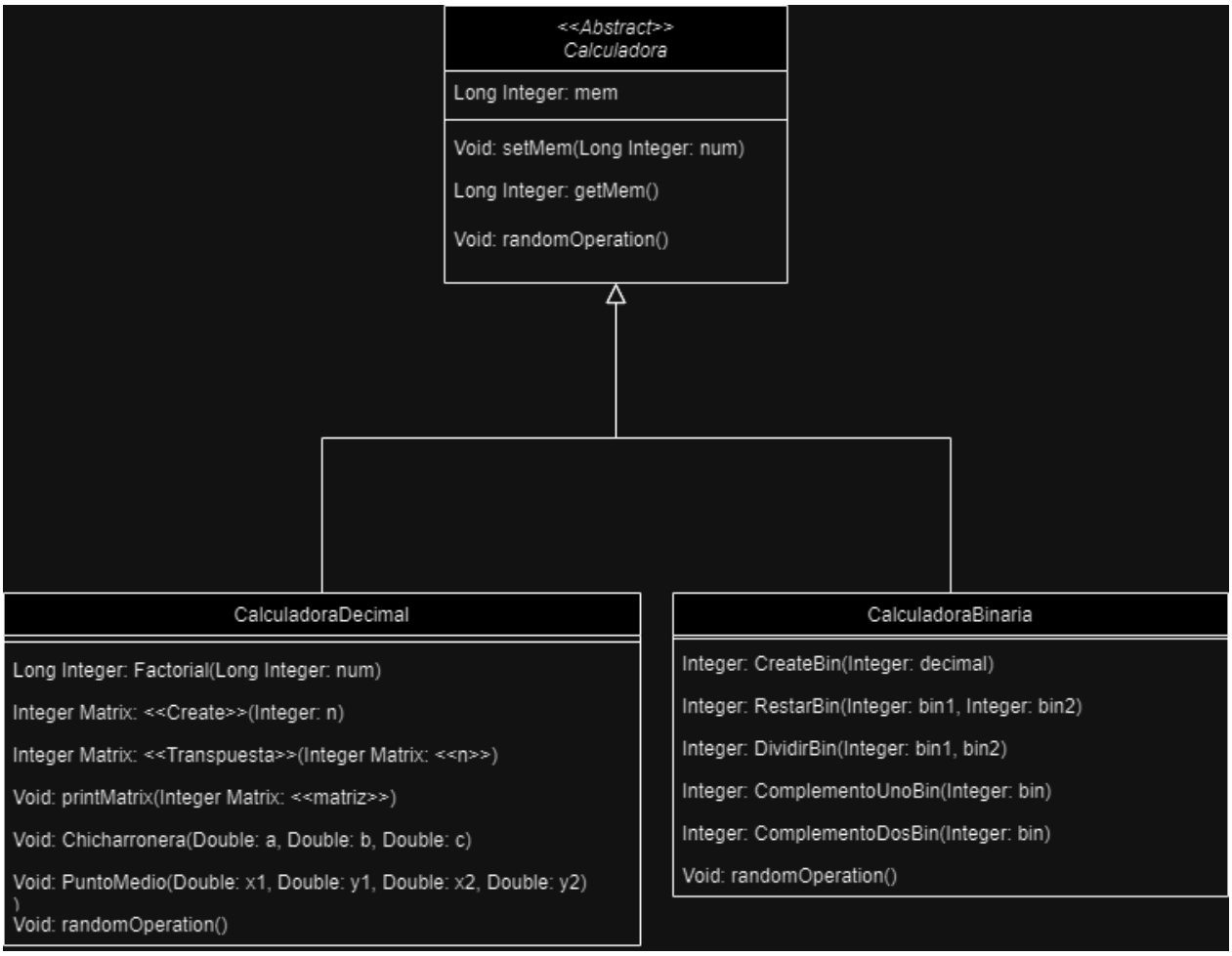

Figure 2: Class diagram for "CalculatorByCheetos"

We have an abstract class for better inheritance handling, as we don't need to declare instances of a calculator. We need to specify that it is abstract before the name. Then we declare all father class attributes and methods (builder included).

As the child classes declare a lot of new methods, their blocks are bigger than the father class's one. Also we need to specify the matrix methods with the "Integer Matrix" label and "<<>>" used for arrays.

2.2.- Object diagrams

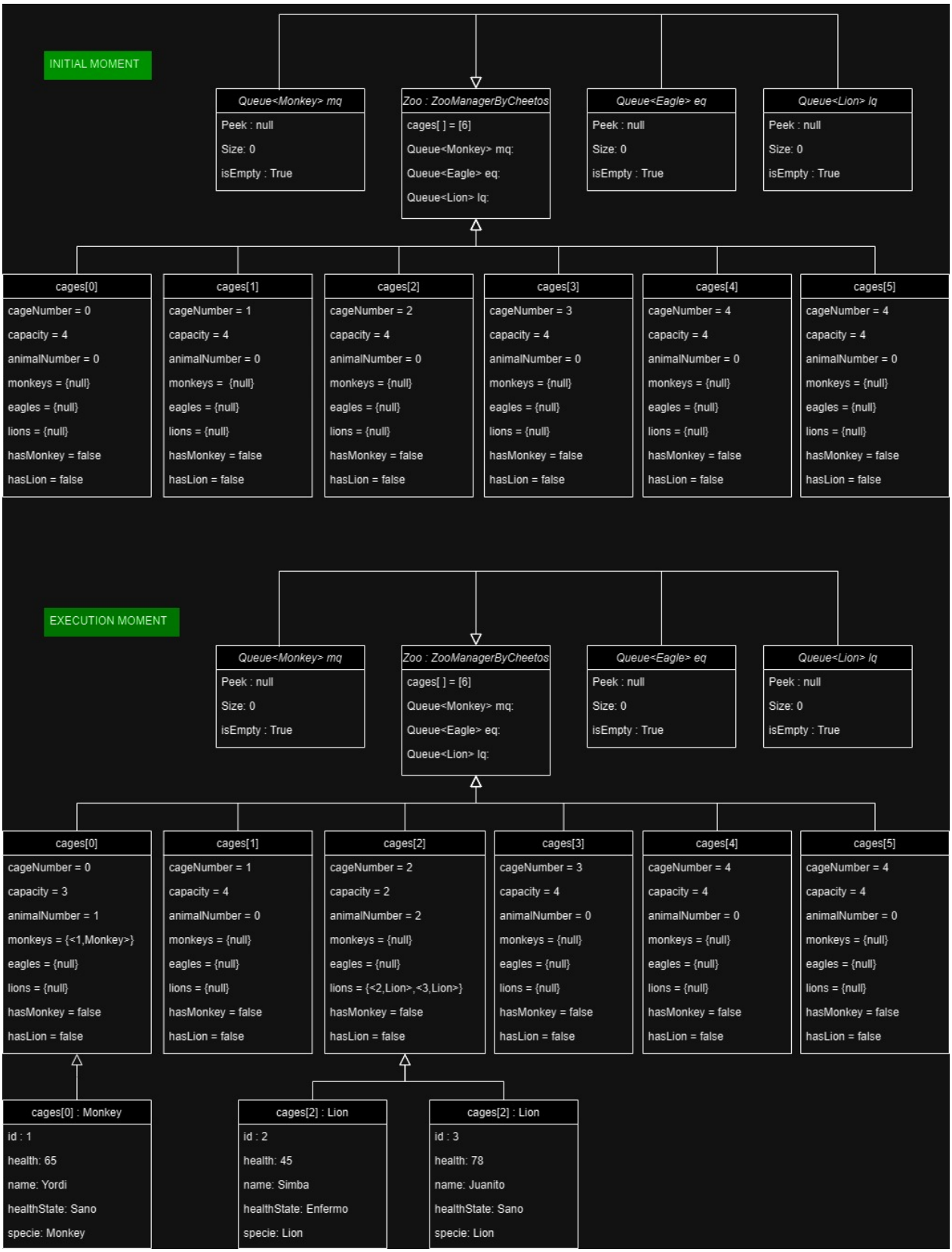

Figure 3: Object diagram for "ZooManagerByCheetos"

As we declare “special” instances such as queues, arrays and hash maps, we need to put various blocks with the attributes and status of those language classes, which makes the diagram bigger.

We chose a random moment by running the program one time making random moves.

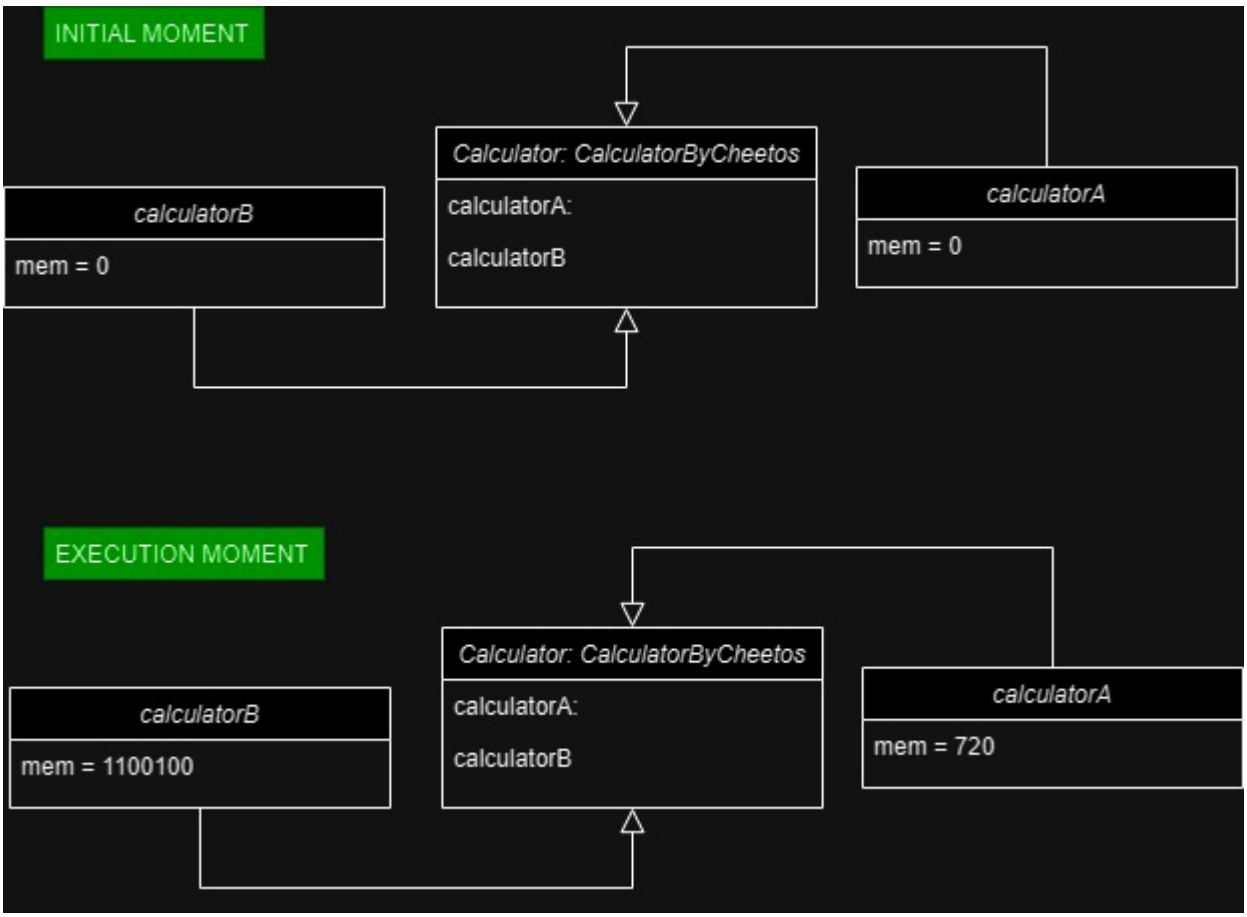

Figure 4: Object diagram for “CalculatorByCheetos”

This is a simpler diagram because we use only one instance of each class, and the classes only have one attribute, the change is very minimal between the initial and the execution moment.

2.3.- States diagrams.

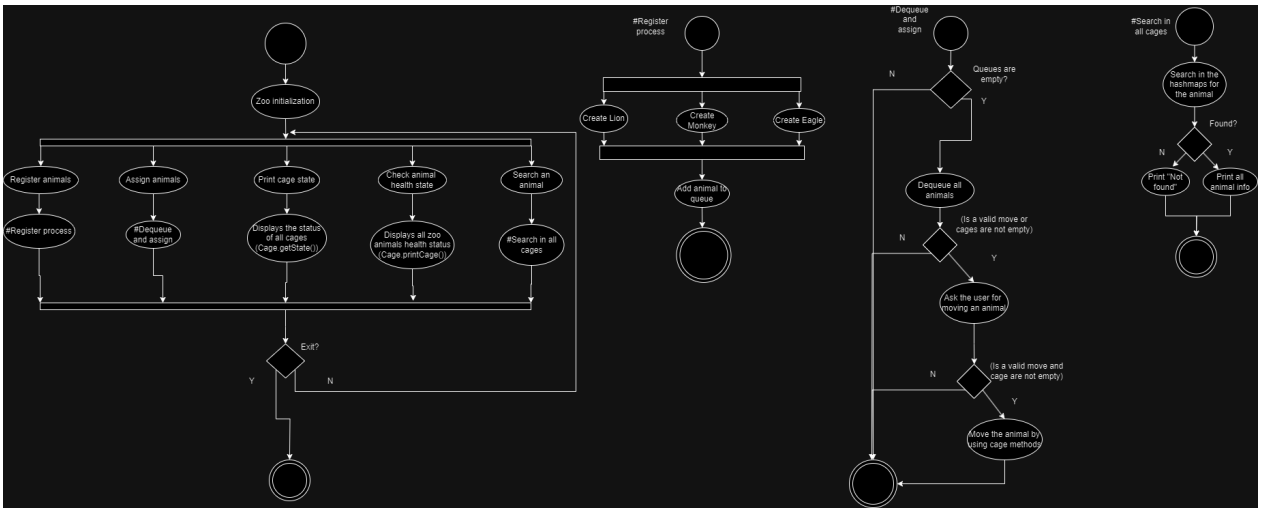

Figure 5: States diagram for "ZooManagerByCheetos"

This diagram specifies all the flow of our program actions. We used all control and decision figures for representing menus and if-else statements for user interactions. The complexity of this program is not that high, but due to possibilities of choosing various scenarios, the state diagram grows a lot.

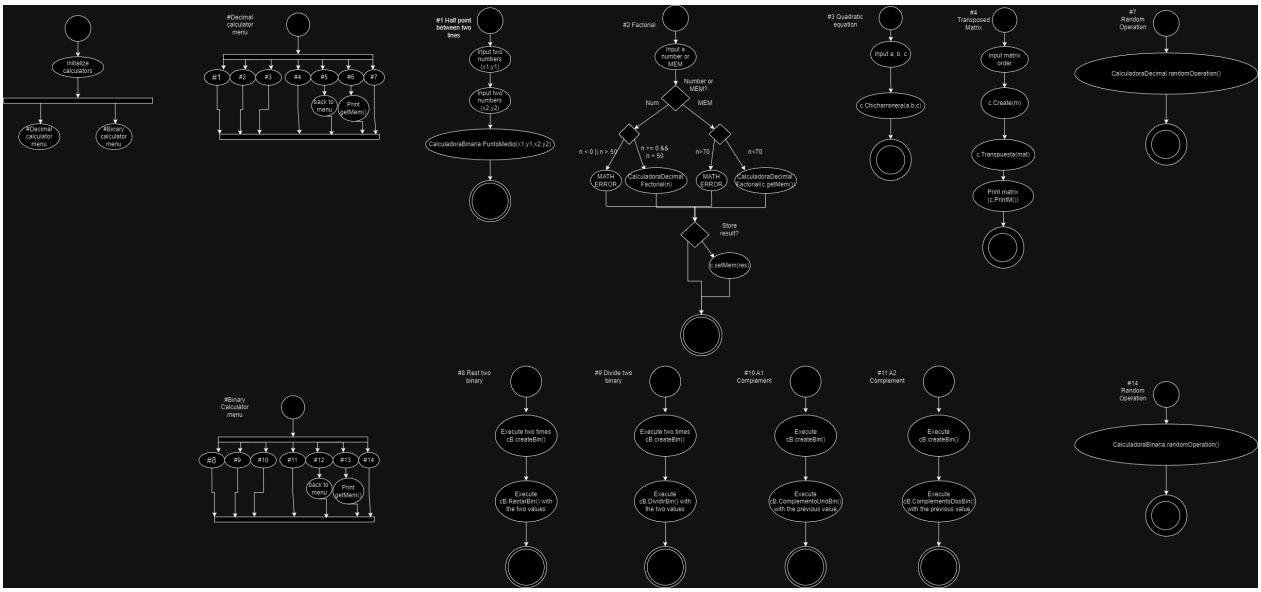

Figure 6: States diagram for "CalculatorByCheetos"

Compared with the last diagram, this calculator diagram grew up because we have two menus with the same choices but different uses, and the menu layers were also difficult to represent, the idea to have nested diagrams, because from a menu, splits in two different menus, and that two menus are splitted in seven different options.

2.4.- Sequence diagrams.

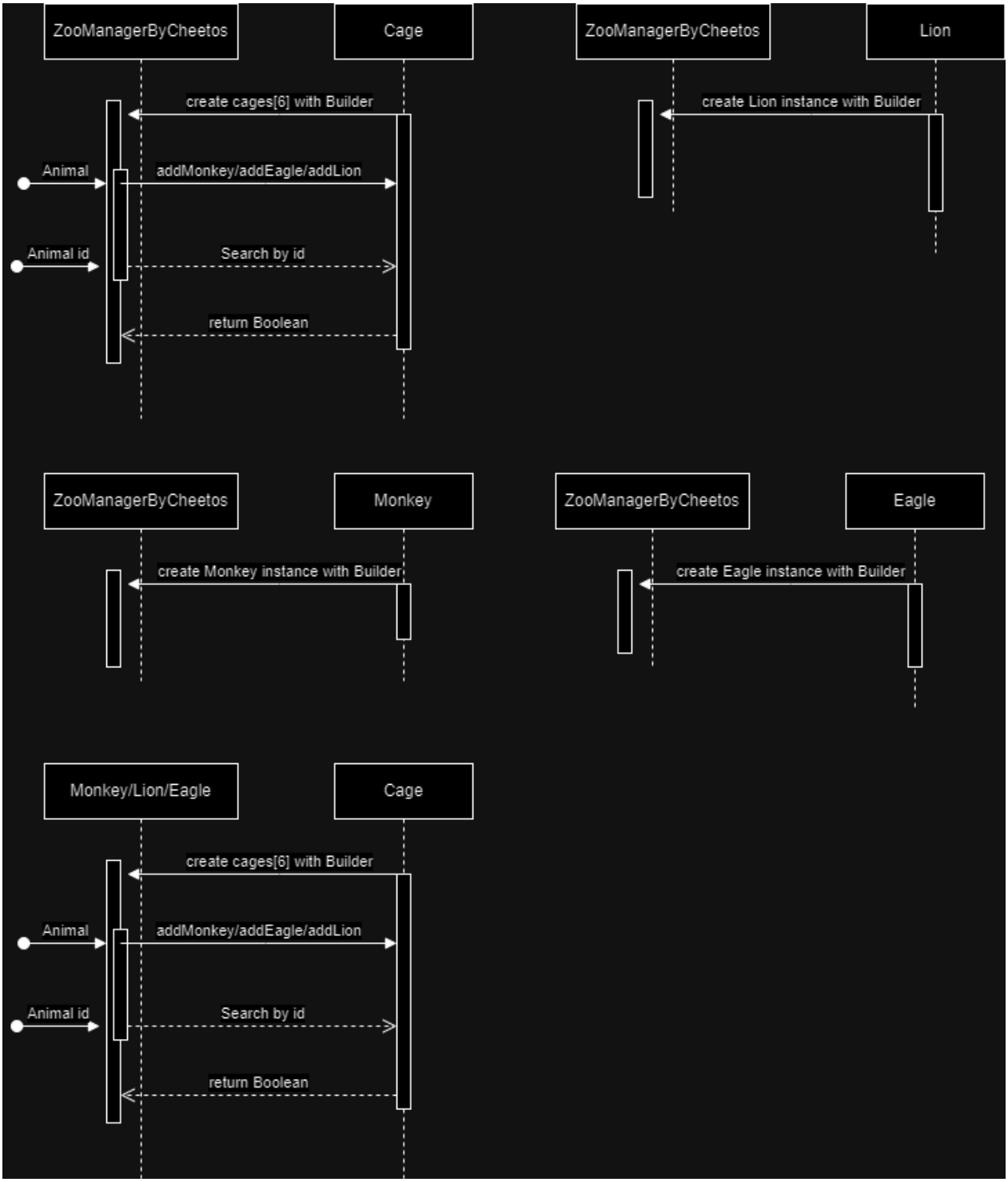

Figure 7: Sequence diagram for "ZooManagerByCheetos"

This diagram extends the description of the runtime functioning of the classes, defining the interactions between classes (main included). As some classes only have limited interaction, there are diagrams with only one arrow pointing to the other class, such as the animal classes.

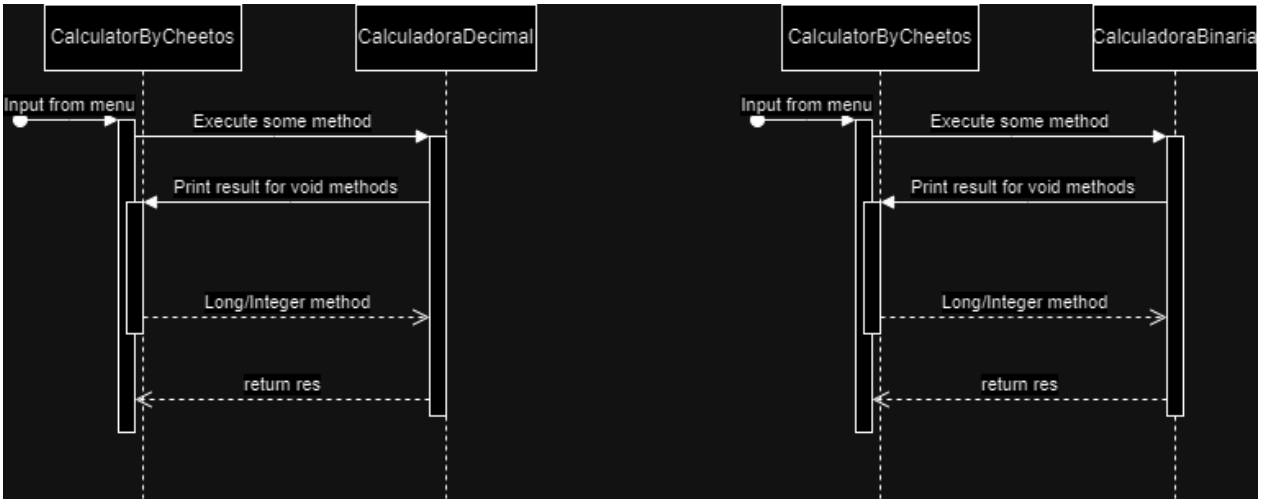

Figure 8: Sequence diagram for "CalculatorByCheetos"

As we have only two instances, we need only two sub diagrams for representing the interactions. Due to the function of the program, the interaction between main and the two classes is very extensive. The two calculator child classes are not interacting in the program, so we don't need a subdiagram for representing that sequence.

Conclusion: The hypothesis was correct, and remarking that making these diagrams is very tedious, but has a lot of utility for code documentation and is crucial for the life cycle of object oriented software, because is easier to understand a system with this kind of diagrams instead of the code, reading code takes a great amount of time, and in some projects we don't dispose of time.

References:

- [1] *Edelmiro Gonzalez Monsivais*. (2013, July). "UML, Lenguaje de Modelado Unificado". Available: https://www.itesrc.edu.mx/portal/articles.php?id_art=1#:~:text=UML%20es%20una%20herramienta%20que,y%20diseño%20de%20un%20problema.