	Carátula para entrega de prácticas
Facultad de Ingeniería	Ingeniería en Computación

Profesor(a): _____

Asignatura: Programación Orientada a Objetos

Grupo: 1

No de Práctica(s): 7

Integrante(s): 11700202-9

_____

_____

_____

No. de Equipo de cómputo empleado: _____

Semestre: 2024-1

Fecha de entrega: 22 de octubre de 2023

Observaciones: _____

_____

CALIFICACIÓN: _____

Index.

Practice 7 - Inheritance and Polymorphism.

Zoo Manager.

1.- Introduction.

This report has the objective of explaining the seventh practice of the Object oriented programming subject, in which we must explore inheritance and polymorphism concepts, typical of the object oriented paradigm.

We must build an algorithm for simulating a zoo manager, in which we must include: Cage changing, health status checker, animal locations, etc.

1.1.- Hypothesis

We have to create an abstract class Animal, because we don't need to declare an instance of this father class, and also gives us for better use of polymorphism without using Overriding.

2.- Development

The steps to follow were:

- Create the cage class.
- Create the animal abstract class.
- Inherit the three child classes needed (Eagle, Monkey and Lion).
- Define the interaction between objects in the main class.

2.1.- Cage class abstraction.

We need eight attributes:

- Animal number (Integer).
- Cage number (Integer).
- Capacity (Integer).
- Monkeys (Hash map (Integer, Monkey)).

- Lions (Hash map (Integer, Lion)).
- Eagles (Hash map (Integer, Eagle)).
- Has Monkey (Boolean).
- Has Lion (Boolean).

We define the next getters:

```
public int getNumber(){return this.cageNumber;}

    public int getAnimalNumber(){return this.animalNumber;}

    public boolean getMonkey(){return this.hasMonkey;}

    public boolean getLion(){return this.hasLion;}

    public Monkey getMonkeyNode(int id){return this.monkeys.get(id);}

    public Lion getLionNode(int id){return this.lions.get(id);}

    public Eagle getEagleNode(int id){return this.eagles.get(id);}
```

Builder:

```
public Cage(int num){

        this.cageNumber = num;

    }
```

Public methods:

addMonkey (void): Validates if it's a valid move using hasLion and hasMonkey, if it's a valid move, then adds the Monkey to the monkey's hash map.

addLion (void): Validates if it's a valid move using hasLion and hasMonkey, if it's a valid move, then adds the Lion to the lion's hash map.

addEagle (void): Validates if it's a valid move, if it's valid, then adds the Eagle to the eagle's hash map.

deleteMonkey (Monkey): Deletes the desired monkey from the monkey's hash map.

deleteLion (Lion): Deletes the desired lion from the lion's hash map.

deleteEagle (Eagle): Deletes the desired eagle from the eagle's hash map.

search (boolean): Searches in the three hash maps for the animal id given, if it's found, returns true, otherwise, returns false.

getState (void): Prints the health status of the sick animals and the amount of different species in the cage.

printCage (void): Prints all the information of the animals contained in the cage.

Class Cage
- Animal number (Integer). - Cage number (Integer). - Capacity (Integer). - Monkeys (Hash map (Integer, Monkey)). - Lions (Hash map (Integer, Lion)). - Eagles (Hash map (Integer, Eagle)). - Has Monkey (Boolean). - Has Lion (Boolean).
- **addLion (void)** - **addEagle (void)** - **deleteMonkey (Monkey)** - **deleteLion (Lion)** - **deleteEagle (Eagle)** - **search (boolean)** - **getState (void)** - **printCage (void)**

2.2.- Animal class

We need five attributes:

- Animal id (Integer).
- Health (Integer).
- Name (String).
- Health State (String).
- Species (String).

We define the next getters:

```
public String getName(){return this.name;}

    public int getId(){return this.id;}

    public int getHealth(){return this.health;}

    public String getHealthState(){return this.healthState;}

    public String getSpecies(){return this.species;}
```

And the next setter:

```
public void setSpecies(String species){this.species = species;}
```

Builder:

```
public Animal(String n, int id){

        this.name = n;

        this.id = id;

        int max = 100;

        int min = 1;

        int range = max - min + 1;

        int x =(int)(Math.random() * range) + min;
```

```
        this.health = x;

        if(this.health > 50){

            this.healthState = "Sano";

        }else{

            this.healthState = "Enfermo";

        }

    }
```

We need only one method, and it's optional.

Sound (Abstract, void): Prints the sound made by the respective animal

<<Abstract>> **Class Animal**
- Animal id (Integer). - Health (Integer). - Name (String). - Health State (String). - Species (String).
- **Sound (Abstract, void)**

2.2.1.- Inherited classes

In the inherited classes we only define the animal sound method and include the specie in the builder.

Monkey:

```
package zoo.animals.terrestrial;
```

```java
import zoo.animals.Animal;


public class Monkey extends Animal{

    public Monkey(String s, int id){

        super(s, id);

        this.setSpecies("Mono");

    }


    public void sound(){

        System.out.println(this.getName() +": Ohhh ohhh ahhh ahhh");

    }

}
```

Lion:

```java
package zoo.animals.terrestrial;

import zoo.animals.Animal;


public class Lion extends Animal{

    public Lion(String s, int id){

        super(s, id);

        this.setSpecies("Leon");

    }


    public void sound(){

        System.out.println(this.getName() +": RAAAAAAAWWWWWWWWRRRRRR");
```

```
    }

}
```

Eagle:

```
package zoo.animals.aerial;

import zoo.animals.Animal;


public class Eagle extends Animal{

    public Eagle(String s, int id){

        super(s, id);

        this.setSpecies("Aguila");

    }


    public void sound(){

        System.out.println(this.getName() +": Eaaaaah eaaaaaah!");

    }

}
```

2.3.- Main class.

It starts by importing various classes that are part of a zoo management system, including classes for animals (monkeys, lions, eagles), cages, and Java utilities.

It sets up several data structures, such as arrays for cages and queues for different types of animals (monkeys, lions, eagles).

The program enters a menu-based loop where you can choose from several options:

a. Register an animal (monkeys, lions, or eagles) by providing a name. The program assigns each animal a unique ID.

b. Automatically assign registered animals to cages based on certain rules, and then display the status of the cages.

c. View the status of the cages, including which animals are in each one.

d. View the health status of animals (not fully implemented in the provided code).

e. Search for the location of an animal based on its ID.

f. Exit the program.

In the "Assign animals to cages" option, the program automatically assigns animals to cages based on their type (monkeys, lions, eagles) and ensures that each cage has a limit of 4 animals.

If you choose to change an animal's cage, you can select a destination cage, move the animal there, and update the cage's status

The program keeps running until you choose to exit it, and it closes the scanner for user input.

Reasons about using the Queue and HashMap data structures.

HashMap: Allows a comfortable search method in O(1) time, avoiding running linear search in arrays.

Queue: If the user wants to define various animals, they can wait until assignation on the queue data structure, also it allows us to assign a great quantity of animals in one execution of the second case.

Conclusion: The hypothesis was correct, the abstract classes made the program very inuitive in terms of handling the OOP paradigm. In this time the menus also were tedious to program but I think we handled correctly all the cases at the main class and at the objects implementation.

References:

- [1] Oracle. (2023) Class LinkedList<E> [Online] Available: https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html
- [2] Oracle. (2023) Interface Queue<E> [Online] Available: https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html
- [3] Oracle. (2023) Class HashMap<K,V> [Online] Available: https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html