cct | College of Computing Technology
Faculty of Information Technology
Object Oriented Construct

Application Development Project – CCTZoo

Asmer Bracho – 2016328
Group B
Lecture Mark Morrissey

April 13, 2018

Table of Contents

1. Assignation of Tasks

	Model					View	Controller
	TheModel <<class>>	animals <<Package>>	Keepers <<package>>	SetUp <<class>>	Search <<class>>		
Asmer	X	X	X	X			X
Miguelantonio						X	X
Adelo	X			X	X		X

Table 1. Allocation of roles and responsibilities

2. Role within the Project

2.1. Creation of the MVC packages Structure

The creation of the project has been done with a MVC pattern Design. The Idea of doing this is being able to segregate and separate group tasks in an organized and reusable structure, so by having a Model which decides the behaviour and usability of the code, and a view which is independent of the model. By doing this we achieved two different aspects.

First of all, independent make group work in a sense much easier, cause the 2 parts are going to collide with the controller and make the application work.

And second of all it is an application reusable in terms of design patterns, for instance by having a model that does not depend at all on the view you can redesign in some stage the interface (view) or even apply the model to a totally different interface, And because of the independent of the model, the code will be reusable.

2.2. Developing of the Model

2.2.1. The Problem

Developing a User Interface Application that allow the user deal with data an keep track of registers such as: Animals information and keepers information.

The System should be able to let the user interact in a sense that he/she is able to:

Create Keepers
Update Keepers
Search for Keepers

Create Animals
Update Animals
Search for Animals
The Animals are broken down into types:

Mammal
Avian
Reptile
Insect
Aquatic

NB: Keep in mind some types of animals can be a combination of the types above.

2.2.2. The Solution

The approach to solve this task, will be mainly focus on the facts that all the types of animal share properties and behaviours that can be allocated in a common parent class (inheritance) . Having said that an Abstract Class Animal will be created that contains the common variables for all Animals as well as methods that will be reused for each type of animals. See *Figure 1*

```
/**
 *
 * @author Adelo Vieira (2017276), Asmer Bracho (2016328) and Miguelantonio Guerra (2016324)
 */
public abstract class Animal implements Serializable {

    private static int id; // Static Variable that help to keep track of number without redundance
    private final int exhibitionNumb; // Exhibition Number for each animal
    private String specie; // Specie
    private String name; // Name of the animal
    private String dateOfBirth; // Date of Birth
    private String dateOfArrival; // Date of Arrival to the Zoo
    private String gender; // Male or Female
    private Medication medication; // Object Medication
    private Vacine vacine; // Object Vaccine
    private ArrayList<Animal> offsprings; // a List of Offpring for each Animal

```

Figure 1. Abstract Class Animal

In terms of the Types of animals, Interfaces for each type will be created, and this will be implemented by the classes of that specific animal. *See Figure 2* for details.

```
/**
 *
 * @author Adelo Vieira (2017276), Asmer Bracho (2016328) and Miguelantonio Guerra (2016324)
 */
public class CreateMammalAquatic extends Animal implements Mammal, Aquatic  {

    private int furry;
    private int canBeOutSideWatter;

    /**
     * Constructor for CreateMammalAquatic
     *
```

Figure 2. This class will extend the Animal Class and implements the Mammal and Aquatic Interface.

Why is this Useful?

By having an implementation that decide the type of animal that object is, we can refined our searches in a very simple way, since our Object animal it is an instance of both interfaces Mammal and Aquatic, so theoretically that Animal can be either of those types, and access to the qualities corresponding to that type.

The different with this or creating a type that it just a combination, is that for that case the animal will be a mixed type, so when searching if you search for an individual type (like Aquatic) you will not get MammalAquatic Animals, because this is not an instance on Aquatic it would be just an instance of MammalAquatic.

As a conclusion and given the case that searches are a fundamental for this application Software we have decide to go with the implementations of interfaces.

2.3. Walking through the Model

2.3.1. Animal

As I mentioned in section 1, this is the parent Class that contain all the instances and method that are going to be extended by the other classes responsible for creating specifics types of Animals.

```
private static int id; // Static Variable that help to keep track of number without redundance
private final int exhibitionNumb; // Exhibition Number for each animal
private String specie; // Specie
private String name; // Name of the animal
private String dateOfBirth; // Date of Birth
private String dateOfArrival; // Date of Arrival to the Zoo
private String gender; // Male or Female
private Medication medication; // Object Medication
private Vacine vacine; // Object Vaccine
private ArrayList<Animal> offsprings; // a List of Offpring for each Animal
```

Figure 3. Common Instances for the Animal Class

The Animal Class contains an abstract Method createOffspring. This method takes parameters as follow:

```
public abstract void createOffpring(String name, String dateOfBirth, String dateOfArrival, int gender, Medication medication, Vacine vacine);
```

The idea of this method is that by being abstract will have to be implemented by any Class that extends Animal, and the reason behind is that every child class will have the ability of creating their own child that will be and object of its own. See *Figure 4.*

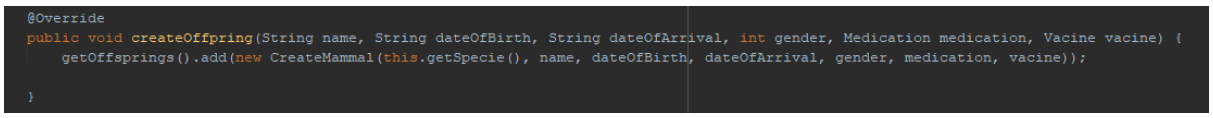

Figure 4. This is a Method within the Class CreateMammal. As can be seen the method createOffsprings is creating an object of the Same Type (which it makes sense, since every child of an animal of type Mammal have to be a mammal). Notice the method signature does not take the parameter specie, but when creating the object the specie is given by the parent specie (pushing the specie to be the same as the parent (which it makes complete sense, and has to be like that)).

2.3.2. Collection of Classes to Create Animals

This collection is constituted for the following classes:

CreateMamal
CreateAvian
CreateReptile
CreateInsect
CreateAquatic
CreateMammalAquatic
CreateReptileAquatic

Each of these classes extends the parent Animal and extend the interface of the corresponding type. Each of the interfaces follow a pattern pretty much the same with the only different that each on then add to their instances an unique property for the type being created (for instance Mammal has the new instance furry, Reptile the new instance isVennon, Avian, canFly and so on).

2.3.3. Data Storage

Having a Software that deal with data, this data should by storage somehow. In this case we will take the approach of serialization to storage the data that we are dealing with.

The process of serialization will consist in the creation of files of extension .ser that are going to be populate with an object which is not other that the List of Arrays that contain the Animals and Keepers when the program is running.

When the program runs for first time a SetUp method will set a random list of Animals and keepers to work with, and immediately the data will be serialized. From then and so on while the file created still exist in the directory, a process of retrieving the information from the files to the system will be performed (deserialization).

```
public void serialization() {

        FileOutputStream fileOut = null;
    try {
        // we will need:
        // 1) FileOutputStream
        fileOut = new FileOutputStream(file);
        FileOutputStream fileOut2 = new FileOutputStream(file2);
        // 2) ObjectOutputStream - and pass on the FileOutputStream
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        ObjectOutputStream out2 = new ObjectOutputStream(fileOut2);
        // now we will need to write the object
        out.writeObject(listAnimals);
        out.flush();
        // Close file
```

Figure 5. Part of the serialization Method

2.3.4. SetUp

The SetUp is a Class that contain a collection of arrays with data that are going to be used to set random list of Medication, Vaccines, Animals and Keepers. *(MVAK)*

The class contains as well a collection of method that will set the data for each of the object mentions above *(MVAK).*

The Medication and vaccines will be set first, then each Animal will take this two object along whit other parameters and finally the keeper will be set taking animals, names and qualifications.

2.3.5. Keeper

This Class create an object Keeper with contains a list of Qualification and a list of Animals that are allow to be looked after.

2.3.6. Collection of Interfaces

The interfaces as explained before will be basically a way of given each animal created a signature that identify them as instances of specifics types (*see Figure 2*).

The interfaces presented are:

Mammal
Avian
Aquatic
Insect
Reptile