COLLEGE OF COMPUTING TECHNOLOGY - DUBLIN
BACHELOR OF SCIENCE IN INFORMATION TECHNOLOGY

OBJECT ORIENTED CONSTRUCTS

Assessment 2 – CCTZoo

Adelo Vieira
Student Number: 2017279
Lecturer: Mr. Mark Morrissey
May 13, 2018

Contents

List of Figures

My main participation in the project was in the programming of:

- the *SetUp «class»*.
- the *Search «class»*.
- the *TheModel «class»*.

1 *SetUp «class»*

This class arises from the need of create a set of data in order to run the system on test data before the Zoo will accept it. Therefore, we have to create a set of random Animal and Keeper objects and put them into an ArrayList.

```
public class TheModel {
    .
    .
    .

    SetUp setup;

    ArrayList<Animal> listAnimals = setup.setListAnimals(200); // Creating at least 200 Animal objects
    ArrayList<Keeper> listKeepers = setup.setListKeepers(50);  // Creating 50 Keeper objects

    .
    .
    .
}
```

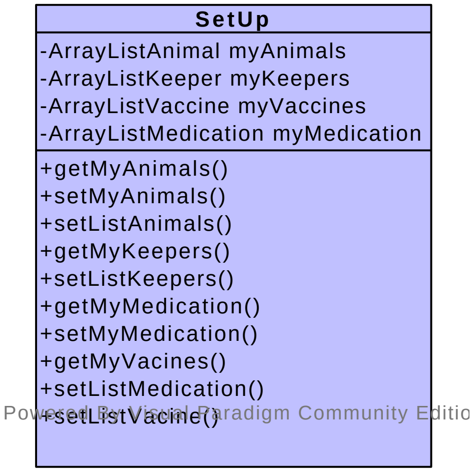
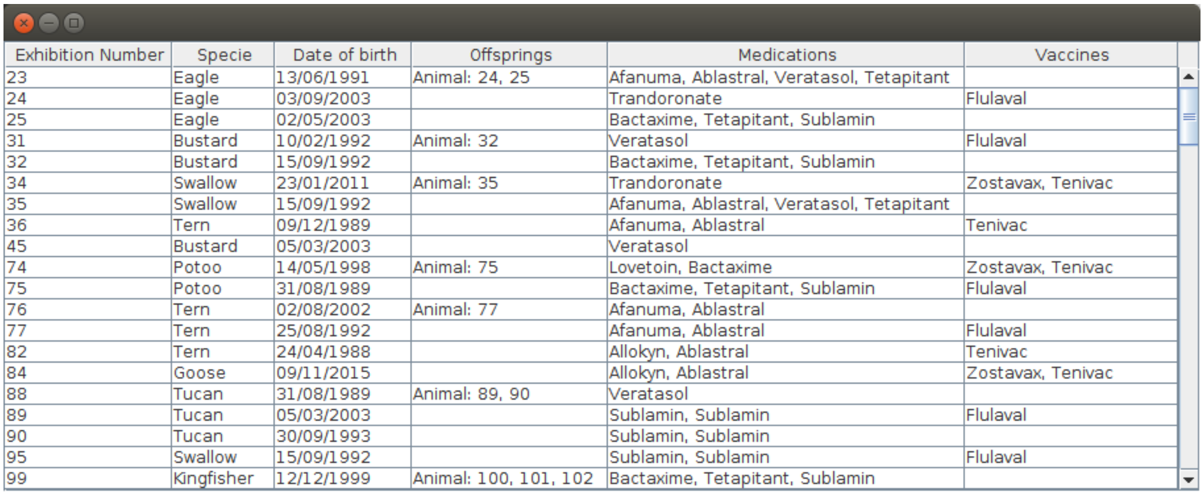


Figure 1.1: SetUp «class» diagram



Exhibition Number	Specie	Date of birth	Offsprings	Medications	Vaccines
23	Eagle	13/06/1991	Animal: 24, 25	Afanuma, Ablastral, Veratasol, Tetapitant	
24	Eagle	03/09/2003		Trandoronate	Flulaval
25	Eagle	02/05/2003		Bactaxime, Tetapitant, Sublamin	
31	Bustard	10/02/1992	Animal: 32	Veratasol	Flulaval
32	Bustard	15/09/1992		Bactaxime, Tetapitant, Sublamin	
34	Swallow	23/01/2011	Animal: 35	Trandoronate	Zostavax, Tenivac
35	Swallow	15/09/1992		Afanuma, Ablastral, Veratasol, Tetapitant	
36	Tern	09/12/1989		Afanuma, Ablastral	Tenivac
45	Bustard	05/03/2003		Veratasol	
74	Potoo	14/05/1998	Animal: 75	Lovetoin, Bactaxime	Zostavax, Tenivac
75	Potoo	31/08/1989		Bactaxime, Tetapitant, Sublamin	Flulaval
76	Tern	02/08/2002	Animal: 77	Afanuma, Ablastral	
77	Tern	25/08/1992		Afanuma, Ablastral	Flulaval
82	Tern	24/04/1988		Allokyn, Ablastral	Tenivac
84	Goose	09/11/2015		Allokyn, Ablastral	Zostavax, Tenivac
88	Tucan	31/08/1989	Animal: 89, 90	Veratasol	
89	Tucan	05/03/2003		Sublamin, Sublamin	Flulaval
90	Tucan	30/09/1993		Sublamin, Sublamin	
95	Swallow	15/09/1992		Sublamin, Sublamin	Flulaval
99	Kingfisher	12/12/1999	Animal: 100, 101, 102	Bactaxime, Tetapitant, Sublamin	

Figure 1.2: Data created in the *SetUp «class»*

My responsibility in this class was in the programming of the method that creates a set of keeper objects (setListKeepers()). Basically, this method generates a set of Keeper objects taking random values of names and *dateOfBirth* from an Array of values previously created.

Another important factor that need to be considered, is that the set of *Animal object* have to be created before creating the *Keeper objects*. This because we need to assign Animals to Keepers (the animals each Keeper look after).

One of the most challenger points in the programming of this method was when assigning the Animals to Keepers. During this step, we had to do many validations in order to make sure that:

- Keepers only get Animals they are qualify to take care of:
    - Because all the animals (no matter its type: Mammal, Aquatic, Reptile, etc) are stored in an ArrayList<Animal>, we had to verify if the object belong to a particular type (using *instanceof*) when checking if an Animal could be assigning to a particular Keeper (given his qualifications).
- The same animal wasn't assigned more that once to the same Keeper.

One of the problems found when checking if an Animal object belong to a child object using *instanceof* was that we had to make a case for each animal type:

```
while (l < noqualify.size() && calificado == true) {
    String noquali = noqualify.get(l);
      switch (noquali) {
    case "Mammal" :
        if(a instanceof Mammal){
        calificado = false;
        }
        break;
    case "Reptile" :
        if(a instanceof Reptile){
        calificado = false;
        }
        break;
    case "Avian" :
        if(a instanceof Avian){
        calificado = false;
        }
        break;
    case "Aquatic" :
        if(a instanceof Aquatic){
        calificado = false;
        }
        break;
    case "Insect" :
        if(a instanceof Insect){
        calificado = false;
        }
        break;
    }
    l++;
}
```

In the above code, the idea was to do something like this:

```
boolean calificado = true;
int l=0;
while (l < noqualify.size() && calificado == true) {
  if (a instanceof noqualify.get(l)){
      calificado = false;
  }
  l++;
}
```

but we found out that in the sentence *«a instanceof noqualify.get(l)»* is not valid because the *instanceof* required directly the name of the *class* we want to check. In our code it wasn't a big deal because we only have 5 classes, but in an case we have a long list of classes our code wouldn't be convenient.

By doing research, we found out that there is another similar operator calls *isInstance()* [tutorialspoint.com] that would allow this kind of operations. However, we could found the solutions. So that would be one of the thing we would like to improve.

2 *Search «class»*

2.1 Search methods

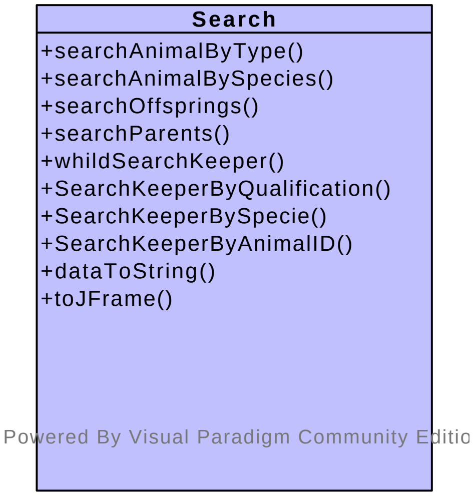


Figure 2.3: Search «class» diagram

This class contains all the method we use to search (or refine) the data. Basically, we created a set of method that take the *ArrayList<Animal>* or *ArrayList<Keeper>* and another parameter related with the kind of search. Let's see an example:

```
// Search offsprings of a particular Animal
public ArrayList<Animal> searchOffsprings(ArrayList<Animal> list, int exhibitionNumb){
    ArrayList<Animal> listSearch = new ArrayList<Animal>();
    for(int i = 0 ; i < list.size() ; i++){
    if(list.get(i).getExhibitionNumb() == exhibitionNumb){
        for(int j = 0 ; j < list.get(i).getOffsprings().size() ; j++){
        listSearch.add(list.get(i).getOffsprings().get(j));
        }

    }
    }
    return listSearch;
}
```

The above code take the *ArrayList<Animal>* that contains all the Animal created in the System and the *exhibitionNumb* of an particular Animal. The method *return* an *ArrayList<Animal>* containing the Animals that are offspring of the Animal represented by the *exhibitionNumb* entered. In the method we make a loop that iterate over the entire ArrayList<Animal> and an *if* that check if the Animal corresponds with the *exhibitionNumb* entered.

All the search method we created follow the same logic.

Now, we realized that this kind of search that we are doing with the methods programed in this class will be easily and robust having all the Animal and keeper data into a Database. That because we could use all the operations implemented with a Database management system.

Therefore, even if we thinks it has been a enriching exercise the implementation of *serializations*, we think the *Version 1.1* of our project should include a *database*

2.2 dataToString metho

This method was implemented in order to simplify the creation of *JTables* for the GUI. Basically, the method takes either the *ArrayList<Animal>* or the *ArrayList<Keeper>* and put all the data in a *String[][]*; which is what a *JTable* takes. The method also display a frame with the *JTable*.

To to it robust, we designed the method to take a String[] of *instanceNames* and a String[] of *columnsNames*. This way, the method return (in the *String[][]*) only the parameters specify for *instanceNames* (in the order entered); and displays (in the *JTable*) these parameters in columns named based on the *columnsNames* array.

```
public String [][]  dataToString(ArrayList<?> listaObjs, String[] instanceNames, String[] columnsNames) {
      .
      .
      .
      if(listaObjs.get(0) instanceof Animal){
      ArrayList<Animal> lista = (ArrayList<Animal>) listaObjs;

      .
      .
      .

      toJFrame(stockArr, columnsNames); // This method displays the JTable

      return stockArr;

      }else{
      ArrayList<Keeper> lista = (ArrayList<Keeper>) listaObjs;

      .
      .
      .

      toJFrame(stockArr, columnsNames);
      return stockArr;
      }
      .
      .
      .
}
```

For example:

```
...

ArrayList<Animal> listAnimals = setup.getMyAnimals();

String[]  instanceNamesA = {"exhibitionNumb",   "specie", "dateOfBirth",   "offsprings", "medication",  "vacine"};
String[]  columnsNamesA  = {"Exhibition Number", "Specie", "Date of birth", "Offsprings", "Medications", "Vaccines" };

// Here we performed a serarh in listAnimals
ArrayList<Animal> searchanimals = search.searchAnimalByType(listAnimals, "Avian");

// Here we convert searchanimals into an String [][]  and display the result  in a JTable (Figure 1.2)
String [][]  searchArray = search.myListObjectToArray(searchAnimalbytype, instanceNamesA, columnsNamesA);

...
```

The result of the above code is shown in Figure 1.2.

Bibliography

CODEJAVA.NET : *Java List Collection Tutorial and Examples.* URL `http://www.codejava.net/java-core/collections/java-list-collection-tutorial-and-examples`.

TUTORIALSPOINT.COM : *isInstance() Method.* URL `https://www.tutorialspoint.com/java/lang/class_isinstance.htm`. 3