Zoo Management App

By John Alexiou

Final project report for

Data Structures and Object-Oriented Programming

420-SF2-RE

Section 00001

Outline

- Project Description (same as deliverable 1)
- Program features
    - How does it meet the requirements
    - Examples
- Challenges
    - Unimplemented features
    - Issues
- Learning outcomes
    - What did I learn from this project

Project Description

- In this zoo management app, Users are stored within a system and play different roles. For example, Visitors will be able to browse a list of various animals, view their characteristics, search for an animal based on a characteristic (using a stream) and view their history of visited animals. On the other hand, zookeepers can also register in the system, and they can browse a list of visitors, visit the most popular animal amongst visitors and even recommend a new animal to add to the zoo. TextIO will be used to write the recommendation on a csv file. A map will be used to show the most popular animal. An interactive menu will be implemented which will allow a User to log in and perform actions based on the type of User(Visitor or Zookeeper).

- In this project, there are two hierarchies: the first one is Animal. Animal class is an abstract superclass which will carry basic animal characteristics,

such as nickname, age, gender and quality (ferocious, shy, clumsy, etc.), as well as a method called makeNoise(), which will apply run-time polymorphism. This method will be overridden in the four animal subclasses, which are Lion, Elephant, Monkey and Penguin. A Comparator class will be implemented inside the Animal class, which will sort the animals by various filters. Next one is abstract User class, which has subclasses Visitor and Zookeeper. User will have attributes name, age and password. IDs for Visitor and Zookeeper will be
different. User will implement an interface called UpdateAccountInterface, which will have two abstract methods: one for updating password and one for updating age. In the Visitors class, a Comparable will be implemented which will sort the visitors by their age.

Program Features

Class hierarchy (2):

Animal à Elephant, Lion, Monkey, Penguin

User à Visitor, Zookeeper

Interface: UpdateAccountInterface with 2 methods: updateAge() & updatePassword()

Polymorphism and method overriding: makeNoise() method in Animal class, being overridden in Elephant, Lion, Monkey & Penguin subclasses.

Data structures used: **List**, used to enclose a variety of animals for the visitors to visit as well as to enclose visitors for the zookeepers to view. **Array**, used in the writeNewAnimalRecommendation() method in Zookeeper class to store the name, gender, age, quality and species of a new animal, and to write it to a csv file. **Set**, used in the searchForAnimalQuality() method in Visitor class to return all animal names who contain the keyword in their quality. The reason why I used a Set is

because every animal nickname in a zoo should be unique, therefore if two animals with the same name share the same quality, it should only display this name once. Finally, a **map** was used in the viewNumberOfVisitorsPerAnimal() method in the Zookeeper class to show the animal name (key) with the number of visitors that animal has (value).

Text I/O: a Filewriter was used to write to a csv file called NewAnimals.csv in the writeNewAnimalRecommendation() method in Zookeeper class. Append = true to write and see multiple animal recommendations.

Stream with Lambda expressions/method referencing: used in the searchForAnimalQuality() method in Visitor class to filter out all animal qualities that do not contain the keyword entered. Lambda expression to filter out the animals whose qualities do not contain the keyword, and method referencing to get the nickname of the remaining animals.

Challenges

Queue and Stack data structures were not used in this project. I felt like List, Map and Set was a better fit for this project, because insertion order was unimportant.

The biggest issue faced was implementing the interactive menu. It was by far the longest part of the project, because it combined methods from both Visitor and Zookeeper class, and used a List of Animals for most methods. Things became messy quick, so keeping things clean and understandable was a challenge that I had to overcome.

Learning Outcomes

Throughout the course of this project, I learnt to manage my time well and to organize my code effectively. Time management was essential for this project, because I had to keep up with all the deliverable due dates while also completing work in other courses. The best way to do this was to spend a bit of time each day on both tasks to ensure that I got both done as effectively as possible. On the other hand, organizing my code was tough, especially for the interactive menu part. So, to keep things clean, I decided to work on one class at a time to ensure that I don't get lost in my code. That way, if my code crashed, I knew where the problem was.