Informatics 1: Object Oriented Programming

Assignment 3 - Report

s2188363

Basic – Design decisions

1. **Animal Package:** It contains an Abstract class called Animal, and Classes for all 8 animals.

   a) **Where:** All 8 classes (Lion.java, Gazzele.java, etc) inherit 2 methods from the Animal Class: getNickname and isCompatibleWith.

   b) **How:** The getNickname method is defined in the abstract Class Animal to not be an abstract method, so all the classes inheriting from Animal can make use of it. However, the isCompatiblewith method is defined to be abstract and is inherited into each of the 8 classes but is overridden by individual definitions.

   c) **Why:** The getNickname method does the same thing for all classes, so it can easily be given a common definition in the Animal class for other classes to inherit and use. However, the isCompatiblewith method must be different for each inheriting class as each animal is compatible in a different way with another.

2. **Area Package:** It contains an interface class IArea, 2 other main classes, 5 other classes representing Areas in the zoo and a helper class to store some data related to a given zoo instance (based on the zoo class in the zoo package).

   a) **Where:** The AllAreaManager and LivingAreaManager classes inherit from IArea and AllAreaManager respectively. Furthermore, Entrance and PicnicArea classes inherit from the AllAreaManager class while Enclosure, Cage, and Aquarium inherit from LivingAreaManager.

   b) **How:** AllAreaManager stores properties required by all 5 areas whereas LivingAreaManager only stores the data and methods common to the 3 habitats like implementation details of AddAnimal, how it is stored etc.

   c) **Why:** This design is followed as there are certain properties that all areas must have while others are limited to only the habitat areas, thereby ensuring all 5 areas inherit only required properties. This helps reduce code duplication, implement abstraction and data hiding.

3. **Zoo Package:** It consists mainly of a final class Codes, an interfere class IZoo and a class called zoo inheriting from IZoo.

   a) **Where:** Class Zoo Inherits from IZoo, overrides all its methods, and makes use of the methods from the StoreArea class defined in Area package.

   b) **How:** Zoo defines an instance of StoreArea to access its methods and data.

   c) **Why:** Zoo majorly calls methods from StoreArea and implements them as the “Controller” or logical block. Store Area was made mainly to act as the Model in MVC from the MVC design pattern, thereby making debugging easier and helping implement abstraction.

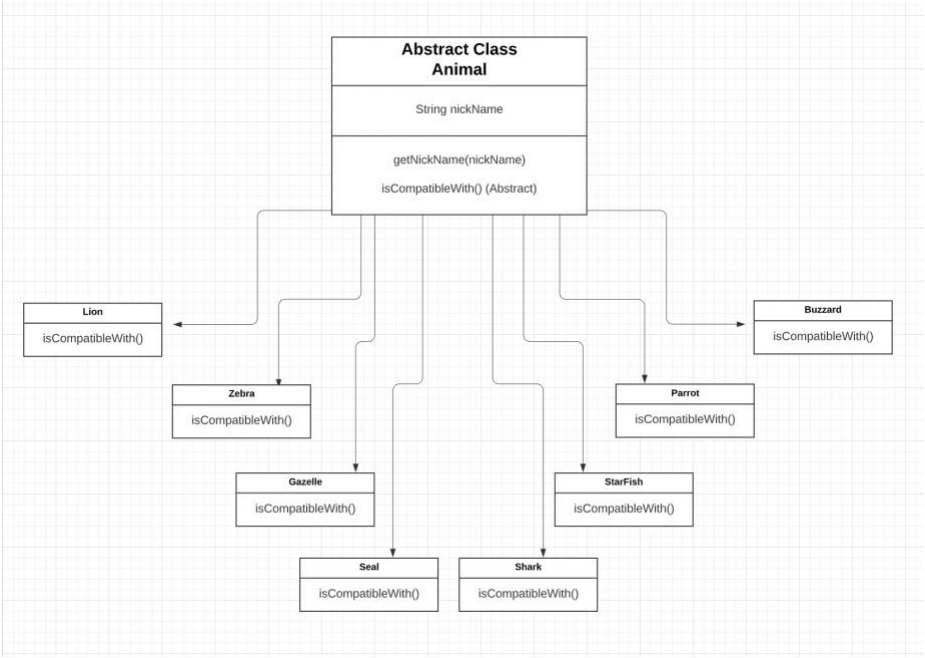

Figure representing the UML Class diagram for animal package

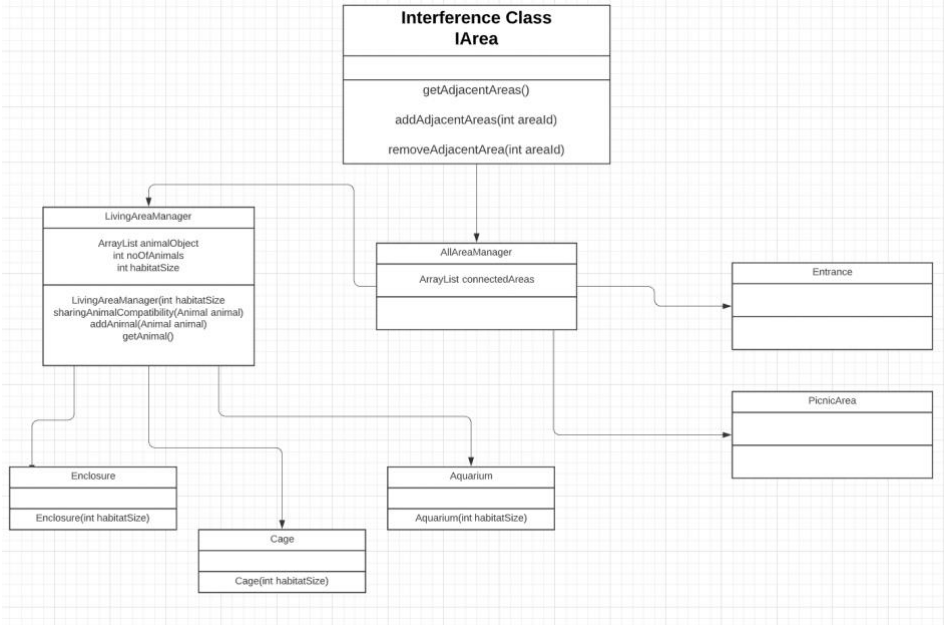

Figure representing the UML Class diagram for areas package

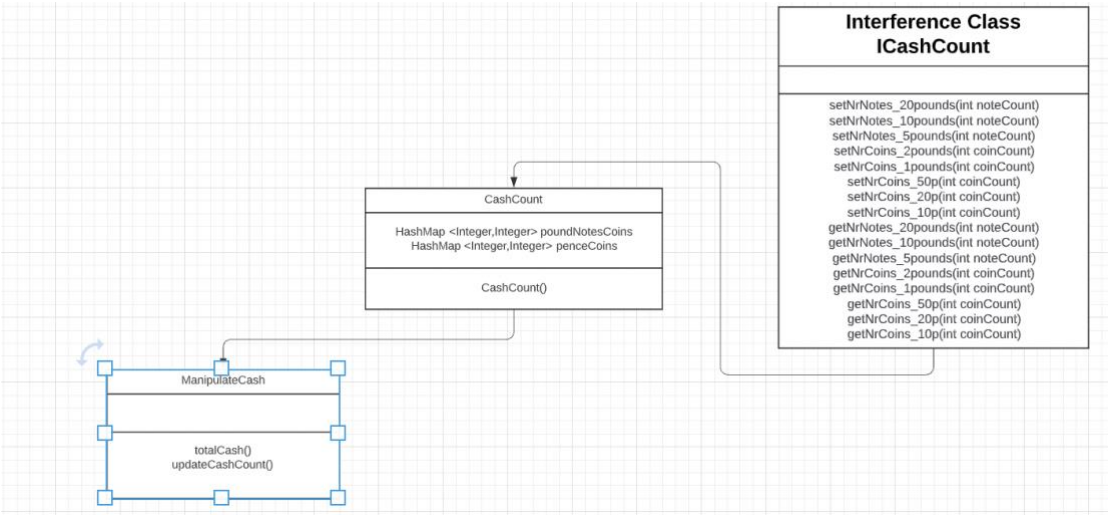

Figure representing the UML Class diagram for the dataStructures package

Inheritance in Zoo Package Can be understood easily without an UML diagram

Basic – Issues encountered

Intermediate – Modelling the zoo's areas and connections

I utilized a HashMap to store the zoo's areas as key value pairs of unique integer ids and instances of the 5 area classes. An Adder, getter and a method to remove areas in the zoo was made to add, remove and return key value pairs from the HashMap, which is stored in another class called StoreArea in the Area package. Thus, I implemented the MVC design (M-StoreArea.java, C-Zoo.java). To connect various areas of the Zoo, I used an arraylist of integers to store the ids of the connected areas. Rather than duplicating code, I made a class called AllAreaManager which implements IArea and defined the required functions to add, remove and get adjacent areas, i.e. adding, removing and returning AreaIds from the arraylist. Eventually all areas inherit from AllAreaManager.

connectAreas method connects the different areas by retrieving the area corresponding to the fromAreaId parameter and adds toAreaId to the Arealist storing adjacent areas for the retrieved instance.

I implemented the IsPathAllowed function, which takes in an arraylist with a potential path for the visitor, checks its validity by looping through the set of given AreaIds and fetching the list of Areaids adjacent to the selected area and then checking if the next AreaId from the set is present in the list.

To check how visitors visit animals in the zoo, the visit method was implemented. It takes an Arraylist containing the areaIds of each area being visited and returns an arraylist of the animals visited. Firstly, I checked whether the path was valid and then proceeded to loop through the list of areaIds; if the corresponding area was a habitat: then I retrieved the arraylist storing the list of animals in that area and added it to another arraylist which is ultimately returned once the loop is finished.

Intermediate – Alternative model

Another way of storing the unique areaIds and their corresponding area instances is by utilising 2 arraylists, one storing the list of areaIds and another storing the list of area instances, which can then be connected by using their indices.

The main problem here is that it leads to longer inefficient code. Furthermore, there are more chances of encountering an error and more exception cases that need to be handled, leading to the code being unnecessarily long. Also, the coding conventions suggest utilising more efficient data structures and making proper use of Java libraries by knowing the API. Another reason is that getters and setters for this implementation would require writing more code whereas these are just one line method calls if we utilize a HashMap as its library is already implemented.

The data structure storing Animals in each area could have easily been an array as the habitat size was predetermined, however arrays containing objects have a default value of null, so when looping through an array storing animals in a habitat like aquarium that is not full, unnecessary nullpointer errors will be encountered, making it better to use an arraylist with habitat size as a separate parameter.

Intermediate – Issues encountered

Issues with findUnreachableAreas :

The major issue I faced in the Intermediate section was while implementing the findUnreachableAreas method in the Zoo class. My approach was to first implement the depth first search algorithm by creating a helper function to retrieve the reachable areaIds and then remove the list of reachable areaIds from the complete set of areaIds to obtain the unreachable areaIds.

While implementing this, my code ended up getting way too long and ran into an unexpected nullpointer errors, this was quickly fixed by a try catch block, assuming that this arose when the graph node hit a dead end (empty ArrayList). Eventhough, this made my program pass the visible intermediate codegrade tests, on further testing I found that it often failed complex text cases indicating some flaw in the logic and my initial assumption.

Later on, I realized that for this particular search problem breadth first search (BFS) is a much quicker and simpler approach with considerably less test cases leading to a simpler and better code. I used an ArrrayDeque as a queue and implemented the algorithm. At the end implementing BFS led to my code running smoothly.

Advanced – Money representation in *ICashCount*

a) **How:** The state of ICashCount's instance represents the money going into the cash machine of the zoo. I utilized 2 Hashmaps in CashCount (which inherits from ICashCount): one storing the pounds and other storing the pence. In each Hashmap I used the value of the note/coin as the key and set the number of denominations as the respective pair.

b) **Why:** Firstly, I had considered using 2 arraylists; storing the denomination in one and their counts in the other. However, that made the implementation messy, long and also had issues with the precision value because of floating points.

   Hashmaps are best solution because pounds and pence can be stored as integers in them. Getters and setters can be one line code because the java library already contains code for Hashmap manipulation. In general, while dealing with 2 related variable quantities it is always best to use Hashmaps.

Advanced – Key ideas behind the chosen algorithm

Advanced – Issues encountered