Informatics 1: Object Oriented Programming

Assignment 3 - Report

<s2101367 Sean Choi>
<16-04-2021>

Overview

- Appendix 1

This assignment is consist of three-part including building a zoo, mapping a zoo, and the ticket machine which will be used for a zoo. Building a zoo is related to a class hierarchy design. An effective class hierarchy design is required. Mapping a zoo problem, subsequently, is about data representation so that the effective usage of data structure is required. Finally, the creation of a ticket machine is accustomed to a search issue. The point of this section is to separate corner cases, so returns values matched with conditions given by the specification.

Basic Features

Basic features require to build a zoo. To build a zoo, several factors that need for zoo are required. A zoo needs animals, and a place for animals, or people. When it comes to adding animals, I used a hierarchy system, which means I divided several classes into two sections. One is for the general features of animals and the other is for specific features of each animal. The Animal class in the animals package is a parent class and the other 8 classes in the same package extend Animal class. The animals have common features such as checking compatibility, checking inhabitation in a certain area, and nickname. But, each animal has different features respectively. Due to that, I used a hierarchy system for avoiding code duplications. 8 kinds of animal are made as each class under the given condition. 8 animals can be divided into three groups based on the given areas. For example, buzzard and parrot is one group. Lion and zebra are other groups. Seal, shark, and starfish are the other group. However, I did not separate these animals into three groups. This is because I think the number of animals class is not that many, and each animal has their features such as compatibility in a certain place.

In terms of designing areas, on the other hand, I created an Area class which is extended by Aquarium, Cage, Enclosure, Entrance, and PicnicArea. Like the Animal class in the animals package, 5 kinds of areas have common features, thus the Area class can be the parent class of these classes. Furthermore, I should have separated 5 kinds of area classes into two groups. One group contains an aquarium, cage, and enclosure. The other group includes Entrance and PicnicArea. Because the latter group does not need to have animals, it would be more effective to

manage classes when there is a group separation. Also, the instruction requires not to modify IArea interface class such that the Area class is required to create a zoo.

Intermediate features

About modelling the zoo's areas and connections

Areas in the zoo used the ArrayList data structure. ArrayList may be an appropriate data structure for creating a large and complex zoo. This representation makes areas to be easily managed. ArrayList is a resizable array so that a programmer can easily insert or delete areas in a particular element from a certain position as much as they want. Furthermore, ArrayList can offer several methods to manage the stored objects such that this can be proper for a zoo that needs a variety of objects such as areas and animals.

In the code related to areas, a zoo roughly has two big concepts; creation area and area connections. Regarding the creation area, ArrayList<Area> areas is a necessary field. This refers to a list of areas. Whenever a programmer needs to call an area created, ArrayList areas will provide an element that indicates a certain area. Also, since all area created is unique, a programmer can assign a unique number to an area by using static int lastID in the field. In the case of an entrance, the zoo can have an entrance with ID zero by setting static final int entranceID = 0 and initialising it at Zoo constructor.

Regarding to area connections, finally, each area needs to know what is an adjacent area around themselves. To figure out it, ArrayList<Area> nextAreas needs to be defined in the field of Area class. The field refers to a list of the adjacent area so that this tells what areas a certain area has adjacently. For example, if we connect area A with area B and area C, then B, C is the adjacent area of area A. In other words, nextAreas of area A contains B and C. By setting a nextAreas, we can easily manage connections between areas.

An alternative data representation

<Hashset vs ArrayList>

When it comes to add, remove, and connect areas, Hashset can be an alternative data representation especially when adding a unique area. This is because Hashset includes unique elements only so that this may be the best way for search operations. However, I used ArrayList since a unique area can also be added, removed, and connected with ArrayList data structure by using static int lastID in the field. Whenever a new area is created, we can create a new id by increasing the global id such that an area can get a unique ID. Therefore, I used ArrayList instead of Hashset.

<Array vs ArrayList>

Array can also be an alternative. However, due to the complexity of creating a zoo, Array may not be appropriate. Array can not change length since the size of the arrays is fixed once created. For example, a new place can not be created once a zoo is created. Also, a new element

can not be inserted or deleted because the elements save in consecutive memory positions so that shifting the whole elements can be costly. This is the same as above when we need to delete a certain place in an array.

Advance Features

Representation of money and its reason

When representing money, two fields need to be defined. The first is to set int entranceFee. By doing that all money can be expressed with one denomination, which is pence. £17.8, for instance, can be expressed in 1780 pence. The second is CashCount cashCount. This field refers to the total money that Zoo has. The CashCount class has a general feature of money such as denomination field and several methods related to money calculation. With the aids of two essential field defined, a programmer can easily calculate or manage money.

Key algorithm idea

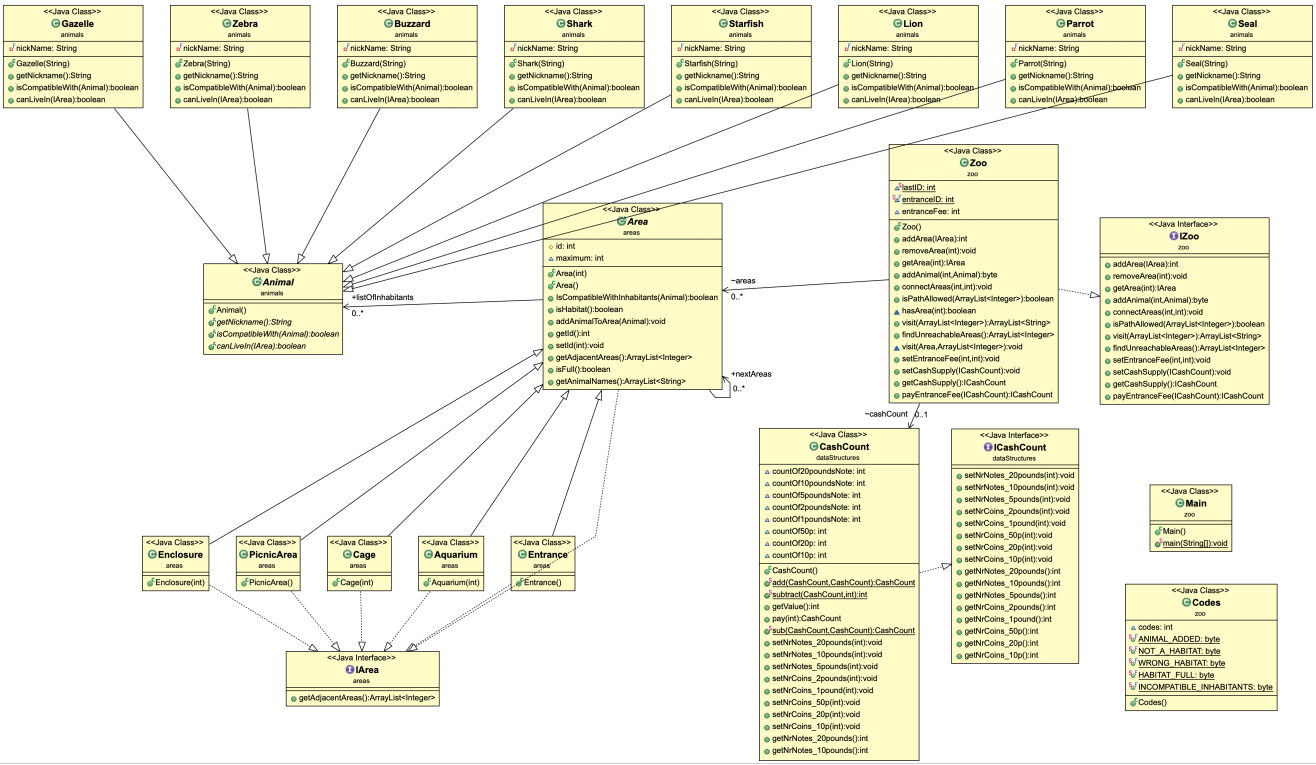
When it comes to key algorithm ideas, my algorithm is consist of a big two axle. One axle is for the easier calculation of money as illustrated above. Unlike setting entrance fee with both int pounds and int pence, we can set this with only one integer pence denomination. The other axle is to convert between one integer type and CashCount type under necessity when calculating change or corner cases. If I elaborate more on that, the payEntranceFee method contains a key algorithm, and the ticket machine needs to return the change after completing corner cases' calculations of the change. Therefore, to return the right change under the instruction, the machine needs to manage corner cases by converting types between one integer type and CashCount type.

As has illustrated in the specification, there are 4 corner cases. For the first case, the machine returns no change when the inserted money is the same as entranceFee. The case does not have a thing to exclude so that only a basic calculation is carried out which is commonly used in all other cases. An inserted money from a visitor needs to be added into total money in the form of each denomination. For instance, if an inserted money is £15, this can be one £5 note and ten £ 1 note, or this can be fifty £1 note. And then, the machine needs to have an integer value of the change by subtracting an inserted money with an entrance fee. Subsequently, update the total money that Zoo has by subtracting the integer change value from it, and then return final the change.

Unlike case 1, the other 3 cases require an additional calculation to return the change. Case 2 refers to the case when an inserted money is smaller than the entrance fee. If an entrance fee is larger than cash inserted, the machine returns the money that was inserted. Next, the machine in case 3 much gives exact change when the inserted cash is too much and the machine can give the change. The key algorithm, in this case, is that the machine returns the change which is supposed to return from a large denomination by using quotient and remainder mathematical concept. In CashCount pay(int value) method, for example, when the machine receives 1200 int value and the machine has £10, £1 notes but they does not have £ 2 note, the machine can return

two £ 1 instead of one £ 2. So the 'pay' method in CashCount class can check the number of a certain note which the machine has and if not enough, the machine hands over the remainder value to the next denomination to return the exact change under the available situation. Furthermore, what if there is no enough the required notes or coins so that the machine cannot return the exact change, they return the cash which was inserted. This is case 4 and a machine needs to restore the original money to the total money in this case.

Appendix 1



[Horizontal Version]

[Vertical Version]