Informatics 1: Object Oriented Programming

Assignment 3 - Report

<s2265910>

Basic – Design decisions

In assignment 3, the basic task is to realize the function of building a zoo. First of all, every zoo needs animals. Although the species of animals are different, they have some common ground. Thus, I created a superclass Animal being inherited from other class. Then, the methods and fields that are the same in each subclass only need to be written in the class Animal such as *getNickname*. If a method is different in each subclass, it will be an abstract class so that its body will be different in each subclass. For example, since different animals have different incompatible animals, the method that test whether it can live with other animals should be different in each specie. In addition, with inheritance, we can give each specie of animal a type initializing in the constructor of each subclass to distinguish the specie of animal.

In order to build a zoo, areas are also important. Since the given *IArea* is an interface and areas do need some methods excluding in *IArea*, we need to create a subclass Area implemented from *IArea* so that areas in the zoo can inherit from *Area*. Thus, with intermediate class Area, we can create areas such as *Cage* by extending the class *Area* and the duplicate methods such as *addAdjacentArea* do not need to be override in subclass like *Aquarium*. In addition, some fields are also needed in subclass could be inherit from class *Area* too. Similar to animal, with inheritance, we can give each area a type initializing in the constructor of each subclass.

After creating areas and animals, we can start to build the zoo. Since the given class *IZoo* is an interface and methods in it do not have body, we need to create a class *Zoo* implemented from *IZoo* and all methods in the *IZoo* should be override. With *IZoo*, we could know the basic function that we need to build and it allow multiply inheritance.

Intermediate – Modelling the zoo’s areas and connections

In order to realize the zoo’s areas and connections, we need to create areas in the zoo first. Since the number of areas is unknown when you first create a zoo, I used array list to store areas in the zoo class and the type in the array list will be class Area. In the zoo, areas excluding entrance and picnic area can add animal into it. To put animal into area, the area should have field to store it, so I create an array list called animals to make sure that the areas can add animal in. Due to a unique id given to each area, it is easier to use unique id to connect each area. Therefore, I also built an array list to store id of all the areas in the zoo. To realize the method *connectAreas*, the areas need to have a method to get its id since the given parameters are id of two areas. Since it is one way system, the areas that can be assessed should be recorded in each area. Thus, I created an array list called adjacentAreas to store the areas that can be reached from current area. Then, to make the connectAreas method realize its function, we just need to add the parameter toAreaId into the adjacentAreas of fromAreaId, so visitors are allowed to go from the area with the fromAreaId to the area with the toAreaId. Since the toAreaId will be stored in the adjacentAreas of fromArea and fromAreaId will not be stored in the adjacentAreas of toArea, visitors cannot go from toArea to fromArea, so it realizes the one-way system. In addition, the connectAreas method will first check whether the fromAreaId and toAreaId are the same to avoid the same parameter. The method isPathAllowed is the function to check whether a list of area can be reached one by one through looking area whether it is stored in the adjacentAreas of the last area.

Intermediate – Alternative model

Using array instead of array list. First of all, compared with array list, array data representation should initialise the size of array first, but you do not know how many areas are there in the zoo when you first create it. Therefore, array could also cannot change the size as the time created. However, array list can add new area into the zoo easily. In addition, this problem will also happen in class area, since adjacent areas and animals in the area are all unknown when you create it. Besides, array list has more useful methods, which have been already written in the package. Thus, when you want to change the array, array list will be much easier than using array. For example, if you want to change an area into the zoo, you only need to use replace method already in the package in array list, but you have to write a loop to change it in the array. In addition, you do not need to know the index of the object that you want in array list, you just need to know the name of object. It is easier for you to modify or get the exact object in your array list.

Advanced – Money representation in *ICashCount*

The money represents in *ICashCount* depends on the denomination. The money will represent the amount of money win terms of the following notes and coins: £20, £10, £5, £2, £1, 50p, 20p, and 10p. You can get the money by knowing the number of notes and coins. This way of representing can make the method of changing money be realized and similar to the real life, since we just have limited denomination. Since it is close to life, it will make the project practical and can be applied in many places. The notes and coins can be easily changed in different region such as dollar and yuan, with changing the denomination in this class. In addition, it will be helpful to greedy algorithm since using integer will make plus and minus easier.

Advanced – Key ideas behind the chosen algorithm

Since the there are four different situations, we need to fix each one and we need to reorder them to see which one should be check first. If the money is less than the price of ticket, the machine does not need to check how much cash should be returned. Another condition is as simple as the situation that the inserted money is less than the price of ticket, which is the inserted money is as the same as the ticket price. In this case, no money will be returned. The hardest task will be the last two situations and they can be fixed together. I try to use the greedy algorithm to solve the way of returning the change, since the instruction want machine to prioritize large denomination. This algorithm will select the largest denomination to be the best option at the moment. It will go through all the denomination from 20 pounds to 10 pence and check one by one. It will record the difference between the inserted money and the ticket price, since the project need this to find the change. Then the difference will be compared with cash supply. To give the visitor large denomination, it needs to try its best at each time comparing with the note or coin. If it has not enough notes or coins at that time, it will give all the notes or coins. For example, when it is checking 20 pounds, assuming that it should return three 20 pounds notes and it only has 2 pounds notes, it will give visitor all the 20 pounds notes. Then the money needed to give back to visitor will reduce to the amount of the reminder from last denomination minus the money given to visitor in current denomination. After checking all the notes and coins in the cash machine, we can know whether the machine can return change for visitor by checking whether there is cash needed to return. If the reminder is not zero, the machine cannot return change for visitor. However, this algorithm will make an issue that the returned cash may not be the least number of notes and coins, which I will issue in the next part.

Advanced – Issues encountered

Issue the payEntranceFee(ICashCount cashInserted)
Since I used greedy algorithm to find the change prioritizing large denomination, this algorithm will make some mistakes. Due to the denomination of 5 pounds and 2 pounds as well as the denomination of 50 pence and 20 pence, the greedy algorithm will pick the wrong 5 pounds or 50 pence when they are checking the number of 5 pounds or 50 pence, which will make the machine get wrong 2 pounds or 20 pence leading to the faulty return. For example, if the ticket fee is 4 pounds, cash supply has one 5 pounds and three 2 pounds, and visitor give one 10 pounds, the greedy algorithm cannot give visitor three pounds since the machine will choose 5 pounds and it will make mistake. I think the instruction could have some misleading or contradiction, since when the machine prioritize large denomination, it might not return the least number of notes and coins. If the project want machine to prioritize large denomination, the greedy algorithm needs to test every case of choosing 5 pounds and 50 pence, since the problem comes from coprime 5 and 2. However, if the project want machine to return the least number of notes and coins, the algorithm of returning change will be totally different. We need to use dynamic programming to find the least number of notes and coins. We need to set two arrays, one for recording minimum coins for p*10 pence and another one for last coin used to make 10*p pence. If we find the change in first array, we will know how many notes or coins needed to use. If we find change in second array, we will know the consist of these notes and coins. This algorithm will eliminate the problem due to denomination of 5 and 2.