Informatics 1: Object Oriented Programming

Assignment 3 - Report

<s2158723 >

Basic – Design decisions

I used inheritance when creating subclasses for all the different animals in the zoo in the 'animals' package, which would inherit the methods of the Animal superclass. It allowed all the animal subclasses to inherit the getNickname() method in the Animals superclass. It helped avoid code duplication as the functionality of the method is general and does not depend on the subclass it is called on. While allowing the functionality of the abstract isCompatibleWith(Animal animal) method to be implemented differently, depending on the subclass of animal used. Since the method performed the same task for each animal but had different criteria, thus a new implementation in each subclass would override the method as needed. I also created two abstract superclasses called Habitats and OtherAreas in the 'areas' package, both implementing the IArea interface. Enclosure, Cage, and Aquarium are all subclasses of Habitats, which also implement the IArea interface. Entrance and PicnicArea are both subclasses of OtherAreas and implement the IArea interface. This hierarchy helps avoid code duplication, as the methods in the IArea interface are the same for all areas in the zoo, therefore creating superclasses that would implement the interface, allows to only write the code in the superclasses and not in each subclass, as the subclasses inherit the methods. The Habitats class also has two other methods, one of which is abstract. The abstract method is implemented in the individual subclasses as they must follow slightly different criteria. The other method (isNotFull(int currentCapacity)) which checks whether a habitat is full or not, is implemented in the Habitats superclass, since it is general for all the subclasses, which yet again, helps avoid code duplication. I chose to create the two superclasses since the Enclosure, Cage, and Aquarium behave similarly and need all the same methods. The same goes for the Entrance and PicnicArea classes. Generally, using inheritance the way I did helped categorize certain classes together, since they behave similarly and must use the same methods, some of which must be implemented in the same way. Without inheritance, I would've had a lot of code duplication, as I would've had to write all the methods in each different class, even though the methods could be reused. It also helped with the implementation of some of the methods in the Zoo class. For example, checking if an area is a habitat was easier, since all the habitats are subclasses of the Habitats class, I only needed to check if the area is an instance of a Habitats class, instead of having to check if it's an instance of the Enclosure, Cage or Aquarium classes.

Intermediate – Modelling the zoo's areas and connections

For the zoo's areas, I used a HashMap, with the IDs as the keys and the areas of type IArea as values. Using a HashMap allows one to easily find any area based on its unique ID; as well as add or remove any of the areas as needed. As for the connections between areas, I used a HashMap, with the unique area IDs as the keys and array lists of area IDs as the values. Thus for every area, there was an array list of the areas it has a connection to, identified through the area IDs. Since the IDs are unique, I used them to help identify areas. For example, when a connection was added from an area with ID 1 to an area with ID 2, 2 would only be added to the array list accessed with the key value of 1. Meaning that there is a connection from 1 to 2, but not from 2 to 1; since 1 is not in the array list accessed by the key 2. I also used a HashMap to store the animals in each area, with the area IDs as the keys and array lists of type Animals as the values. This means that for every area, there is an array list of all the animals in the area. Using HashMaps to store all these different values allowed me to add and remove areas as needed, without any restrictions. So, when an area is added to the zoo, it's added to the areas HashMap, the connections HashMap, and the animals HashMap. The same applies when an area is removed.

Intermediate – Alternative model

The alternative way to store the areas would have been to use a two-dimensional array. However, arrays are rigid; you must choose a set length for the array, which cannot be exceeded. Therefore, there would be a limit to how many areas could be added to the zoo. Whereas, when using a HashMap, a set length is unrequired, and you can add and remove values without any restrictions. Arrays can have duplicated values, whereas a

HashMap can only have unique keys. As the area IDs are unique, using a HashMap ensures that that ID can only be used to identify one area. HashMap also allows the use of custom data types that arrays do not. Therefore, a data type such as IArea or Animal could be used when using a HashMap. For the connections, an array list of array lists could've been used. However, it would've made adding and removing new values more complicated than using a HashMap.

Advanced – Money representation in *ICashCount*

Advanced – Key ideas behind the chosen algorithm

Advanced – Issues encountered