Matthew Trimby c1525379

Improving Zoo/Animal using OO code.

This document will serve to provide a basic overview of how the Zoo and Animal classes could be re-designed to take advantage of the Object-Orientated (OO) nature of java. This would allow for easy expansion of the system in more complicated cases as well as keeping programming at the interface level for the programmer.

Animal.java

To take advantage of the OO aspect of java I would propose making the animal class an abstract superclass. This class would still have features of the current Animal class such as getName() and getAvailableAnimals(), these would be inherited by all subclasses. In this scenario each animal would have its own subclass that extended the animal class - reusing this code meaning that less lines have to be written as each animal simply inherits methods from the superclass.

Another advantage of this method is that the Animal class could have abstract methods that are defined differently in each individual subclass. For example, the feeding times of the lions and the monkeys may be calculated differently - by having an abstract method getFeedingTime() in the Animal superclass and then overriding it in each subclass we allow the method to still be called at the Superclass level even though the process for working out each animal is different. e.g..

- Animal dennis = new Lion()

- dennis.getFeedingTime()

In this way, once all the animal classes are implemented the programmer can program at the interface level because all the references are from the Superclass.

I would also propose that the loan methods of the Animal class were instead stored in an 100% abstract interface “loanable”. The Animal class would implement the loanable interface when it is defined e.g..

- public abstract class Animal implements loanable

Instead of having loanAnimal() and returnAnimal(), the methods for loan() and return() would be overwritten in the Animal class. This means that in the long term the loan method could also be applied to other Objects, other things the zoo may have such as Staff or Equipment could also be loanable to other zoos. These classes would also implement the ‘loanable’ interface even though they would have different processes of carrying out the loan. This massively simplifies the code for the user, instead of having to remember many different method names for the loaning of each individual object, they can simply use the loan() method on any ‘loanable’ object.

Zoo.java

Moving on to the Zoo class I would propose that it was also implemented as an abstract superclass. Each individual type of zoo would then have a class of its own that inherited common attributes/methods of all Zoos such as getting the total number of available animals, but also allowed for each zoo having custom methods. For example some Zoos may be targeted towards

tropical animals or reptiles and may have different needs from from other zoos. By using Zoo as a superclass we re-use a lot of the code whilst still being able to cater for a unique type of Zoos individual requirements.

By making the Zoo class abstract we can give it abstract methods for example getRevenue() that might calculate the revenue of a given Zoo. Although each Zoo's revenue might be calculated in a slightly different way based of the format of the zoo itself, the references are all from the superclass and therefore allow the programmer to program at the interface level.

In theory we could also take advantage of Java's OO nature by creating an abstract interface, "searchable", with methods such as hasWithName() and getWithName(). By abstracting the code in this way, if the system was expanded with other classes such as Staff or Equipment that also implemented the 'searchable' interface, the system becomes much less complicated for the end user as classes share methods of the same name that are implemented differently.