Object Oriented Paradigms in Java

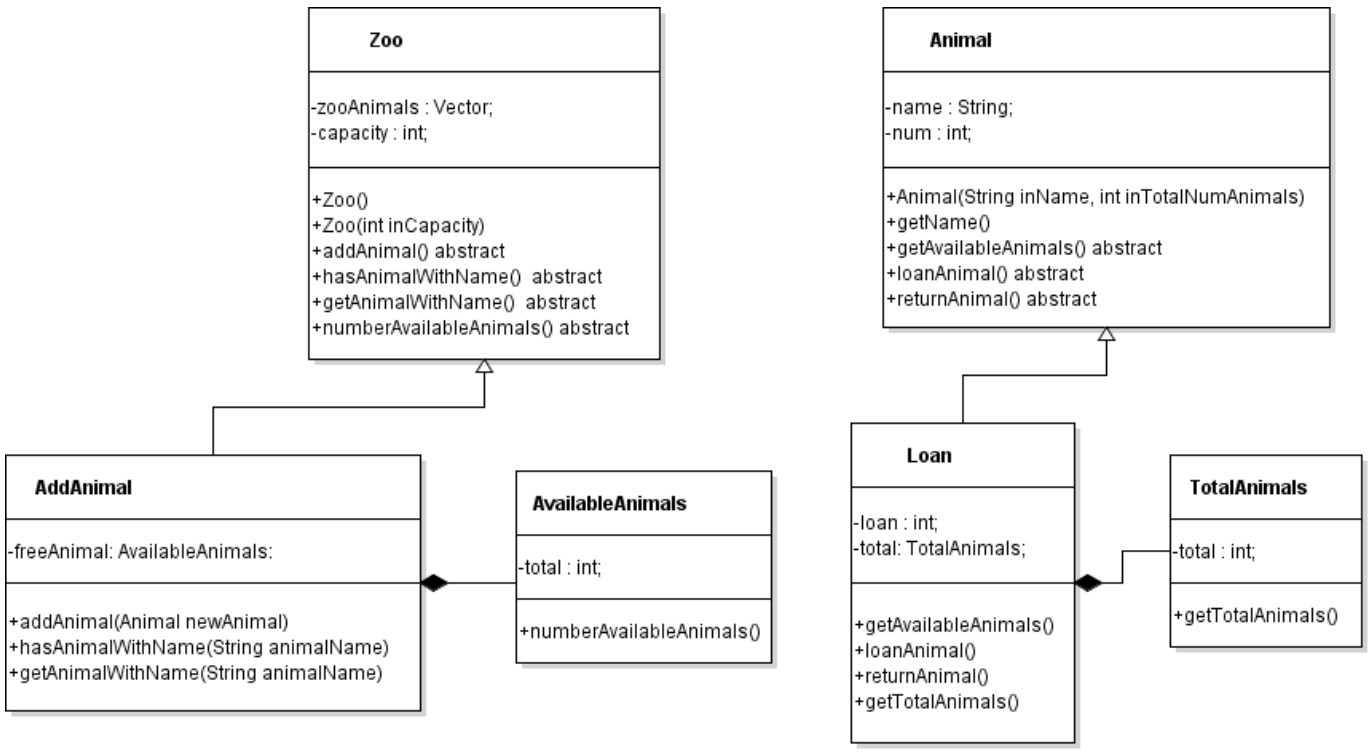
Charles Read (c1646151) - Java Assignment 2: Question 2

Introduction

Object oriented paradigms contain many features and properties that make development and maintainability dramatically more effective. These features, implemented correctly, can allow developers to create encapsulated classes and methods which can be easily identified and reused in different parts of the program to save time as well as additional risk of errors within the unneeded code. The Animal and Zoo classes from question one of this assessment could be redesigned to take greater advantage of the object-oriented paradigm features available in Java. Currently, the program consists of two classes which don't inherit from another class (other than object) and don't fully take advantage of inheritance and polymorphism features within object oriented paradigms.

Redesign

Below is the UML diagram for my redesigned program which splits the classes up into multiple subclasses as well as the two primary super classes. This design allows the two interfaces, Zoo and Animal, to be developed further at an interface level due to the implementation and the interfaces being separate entities as the contents has been encapsulated.



Firstly, the two classes have been kept from the previous version of the program. I decided that it was a good idea to keep these two classes as superclass as they each run separate operations within the program that work well being independent interfaces. The Animal superclass now has a subclass called ‘Loan’, within this class contains all the methods which allow the animals to be loaned as well as returned. The method ‘getAvailableAnimals()’ is also defined in this class.

These three methods are inherited from the Animal superclass but are defined again as they will be upcasted to the abstract methods in the Animal superclass to overwrite the variables and data. This means they will be passed as a reference to the Animal Class with their specific method data only being used from the ‘Loan’ class (reference passes specific variables and not whole class/methods). The Animal superclass is an abstract class, which means it contains one or more abstract methods which cannot be invoked they don’t contain any implementation individually.

The ‘TotalAnimals’ class is a subclass of the ‘Loan’ Class which is inherited by composition, meaning it has been formed from existing classes. This small class contains the integer for the total amount of animals which is then used by the ‘Loan’ class to be returned for use in the ‘getTotalAnimals’ method. The use of polymorphism and inheritance in these classes allows the Animal class to use as an interface for future development, maintenance and expansion. The Animal class now only contains abstract methods as well as a constructor (every class needs their own constructors unless specified otherwise).

Additionally, the Zoo superclass has a similar setup as a subclass named ‘AddAnimal’ now exists which contains methods related to adding animals (using Animal.java). Again, using composition inheritance, the total integer for finding numberAvailableAnimals() is used by the AddAnimal class from the AvailableAnimals subclass.

The subclasses inherit all variables and methods from the superclass, this means that they don’t need to be declared again, thus allowing large amounts of code, in the best case, to only be written once which leads to the reduced chance in errors causing as well as smaller file sizes and in many cases shorter development time. These subclasses also allow methods and variables to be hidden away in their own area, so once completed, they can be tested and implemented into the system much like a functional piece of a puzzle.