ECE 325 Assignment 3: Playing your band’s first gig/show!

--Jaskeerat Singh

3) Why are the exceptions thrown by the ZooAnimalHandler class and not somewhere else? Use the concept of responsibility to explain your answer.

The exceptions are thrown by the ZooAnimalHandler class and not somewhere else because the ZooAnimalHandler class is responsible for all the interactions between the Zoo Animals, the Animal Handler and the Dance Therapist. This is a class that manages the selection of the animals by the handlers to be fed and the animals to be invited to dance and makes sure that the zoo animals only perform tasks assigned to them based on these checks enforced by the exceptions. This makes the code more intuitive and easier to follow, along with modularity for further future expansion.

4) Why we cannot define a private abstract method? Explain your answer.

We cannot define a private abstract method because that would be contradictive in nature. Abstract methods are sort of like templates (method signature) that are defined so that all subclasses to the abstract class must have an implementation of that method (in an overridden format if that subclass is not abstract itself). On the other hand, private methods can only be accessed within the class they have been created and hence cannot be overridden or even accessed in any other class. Both functionalities contradict each other and hence cannot be used collectively in one method.

5) What does it mean that ZooAnimalHandler implements two interfaces? How can we interpret this in natural language?

ZooAnimalHandler implementing two interfaces means that it is implementing all the abstract methods defined in both interfaces. In this case, ZooAnimalHandler has decided to take on the responsibilities of both DanceTherapist (inviting random animals to dance, making sure it is safe to do so) AND AnimalHandler (feeding the animals only if they have danced AND have not eaten already). It must handle both the interfaces tasks.

6) You may have noticed that when implementing the methods that throw an exception in the interfaces, your implementation does not necessarily have to throw an exception. Explain why not.

It is not necessary for the exceptions declared in the interfaces to be thrown while implementing the methods from the interfaces. This is because it is not always beneficial to throw the exception. For example in our feedRandomAnimal method, we are not directly throwing exceptions since there is another method feed(ZooAnimal a) doing it for us. Similarly, there are cases where we know that the exceptions we have declared in the interfaces will simply not rise and hence the methods implementing the interface do not necessarily have to throw exceptions either. This is similar to unchecked exceptions which are also not required to be thrown or handled explicitly in java.