1. Recall from Project 1 the OO program you or your team wrote in Java 8 or later that implements a Zoo full of Animals. Using that code and its original requirements as a basis (see the previous assignment details if needed), extend your Java code to perform the following additional tasks:

a) Add at least one implementation of the strategy pattern by having at least one Animal behavior delegated and referenced by Animals rather than being inherited and overridden. This means when Animals are instantiated, the behavior they need will have to be initialized for them in a strategy pattern manner. Clearly document in the code where the strategy pattern is applied. You may use strategy more than once if you wish.

b) Add an implementation of the observer pattern. Create a new observer class called the ZooAnnouncer. At the start of the program, the ZooAnnouncer (who IS-A ZooEmployee) will arrive at the Zoo each day. The ZooAnnouncer will observe the ZooKeeper, which will be modified to be observable. When the ZooKeeper is starting to perform one of their tasks, they will create an observable event for the ZooAnnouncer. The ZooAnnouncer will respond to this by telling the Zoo (issuing a print statement) something like "Hi, this is the Zoo Announcer. The Zookeeper is about to wake the animals!". (Note, this should not replace the text messages the ZooKeeper themselves may issue from the original flow.) Once there are no further events to announce for the day, the ZooAnnouncer should cease observing and leave for the day. Clearly document in the code where the observer pattern is applied.

c) Add an object to track time at the Zoo called a ZooClock (the ZooClock class is NOT a ZooEmployee). The Zoo will be open from 8 AM to 8 PM each day, and the ZooClock will announce the time in one hour increments and will maintain the current time increment. The ZooKeeper must perform their tasks (wake the animals, roll call the animals, feed the animals, exercise the animals, and tell the animals to sleep) once per day at a time of your choosing. So for instance, if the clock changes to 9 AM, you might choose that as the time the ZooKeeper will wake the animals.

d) Add a new class called the ZooFoodServer, which IS-A ZooEmployee. An instance of the ZooFoodServer will arrive at the Zoo at the start of a day. They will make food, serve food (at 12 PM), clean, make food, serve food (at 5 PM), clean, and then leave for the day, each of which should be noted with a print statement (e.g. "ZooFoodServer is making food."). The ZooAnnouncer will also observe the ZooFoodServer, but will only announce when lunch is served and when dinner is served, not other tasks. The ZooFoodServer will also do their tasks as listed at times of your choosing (other than the two times specified for serving food).

Code should be clearly designed in an OO fashion throughout and should be clearly documented. Full output from the program should be captured and provided as in Project 1.

2. Create a complete UML class diagram for the revised Zoo Program with strategy and observer. The class diagram should be detailed, with all attributes, accessibility, and methods documented, as well as any requirements for multiplicity in associations (e.g. two of each animal subclass, etc.).

3. Create a complete UML sequence diagram for a typical day for the revised Zoo Program with strategy and observer. This diagram should show all top-level objects (:ZooAnnouncer, :ZooKeeper, :ZooClock, :ZooFoodServer, :Animal) that are instantiated, executed, and deconstructed in the lifetime of the code.

To make this readable, do not go to the subclass level; so Animal, yes, but probably not Canine or Dog) and the interactions between those objects.

4. Create a complete UML activity diagram for the revised Zoo Program, showing all major operations that occur from the beginning to the end of code execution for a single simulated day. (Again, details at the subclass levels are not required.)

5. Create a UML Use Case diagram for the tasks the Zookeeper (the Actor) must perform at the Zoo. Consider any <include> (mandatory/ancillary) or <extend> (optional/conditional) tasks they may need to include in a complete view of the scenario. Use the WAVE rule to guide your use case model.

Homework/Project 2 is worth 75 points – 40 points for the program in item number 1, 10 points each for the UML diagrams in 2, 3, 4; 5 points for UML diagram 5. There is no extra credit element for this assignment.

Grading Rubric:

Question 1: 40 points possible.

- README Documentation 10 points (-1 or -2 for incomplete or missing elements, -10 if not there).
- Execution 10 points (-2 or -4 for missing expected functionality or output).
- Code quality 10 points (-1, -2, -3 for issues including poor commenting, procedural style code, or logic errors; -10 if not an object-oriented solution as requested).
- Proper use of strategy and observer patterns 10 points (-1 or -2 for implementation issues or poor commenting, -5 if a pattern is missing).

Any UML tool (draw.io, etc.) can be used to answer questions 2-5. If done on paper/pencil or whiteboard, please be sure the captured diagrams are readable and clear.

Questions 2-4 will be graded on the completeness of the answer, and the correct use of UML. A solid answer will get 10 points, missing elements or mistakes in UML answers will cost -2 points each, missing parts of answers completely will be -4 points.

Similarly, Question 5 (which should be a simpler answer than 2-4), is worth 5 points, with -1 or -2 for missing elements or mistakes in UML.

Submissions:

Question 1 should be submitted via a GitHub Repository URL, including the code, the output text file, and a README Markdown file with the title of the project, the names of team members**,** any comments on or assumptions made for the development, and any special instructions to run the code. Your OO code should be well structured and commented, and citations and/or URLs should be present for any code taken from external sources. Comments should focus on what is being done in the code, not on how, unless the method in question is unclear or obscure. You may not directly use code developed by other student teams, although you may discuss approaches to the problems, and may wish to credit other teams (or class staff) for ideas or direction.

Questions 2-5 must be submitted separately via a PDF file and must include all names of team members.

Homework/Project 2 is Due at 12 noon on Wednesday 9/30.

Assignments will be accepted late as follows. There is no late penalty within 4 hours of the due date/time. In the next 44 hours, the penalty for a late submission is 5%. In the next 48 hours after that the late penalty increases to 15% of the grade. After that point, assignments will not be accepted.

Late penalties will be waived for health or medical related issues (of course).

Use office hours, e-mail, or Piazza to reach the class staff regarding homework/project questions.