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), perform the following additional tasks:

a) Completely rewrite the Zoo/Animal program in another OO language of your choice; any OO language other than Java. The only requirement is that the language does have clear OO support (so, not C or FORTRAN, for instance).

b) 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.

c) Add an implementation of the observer pattern. Create a new observer class called the ZooAnnouncer. At the start of the program, 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 the ZooKeeper completes his last task for the day, the ZooAnnouncer should cease observing and deconstruct. Clearly document in the code where the observer pattern is applied.

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 the revised Zoo Program with strategy and observer. This diagram should show all top-level objects 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. (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 outfile, and a README Markdown file with the title of the project, the names of team members, **the language and environment used for development,** 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's 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. Cite any code from third-party sources or examples.

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

Homework/Project 2 is Due at 12 noon on Friday 2/14.

Assignments will be accepted late for one week. There is no late penalty within 4 hours of the due date/time. In the next 24 hours, the penalty for a late submission is 5%. After that the late penalty increases to 15% of the grade. After the one week point, assignments will not be accepted. The team may submit the assignment using a late pass from each of the team members. Only one late pass can be submitted per project, and it will extend the due date/time 24 hours.

Use e-mail or Piazza to reach the class staff regarding homework questions.