Project Part 6: Final Report & Code Completion

1. *Name*
   Courtney Solano

2. *Project Description*
   Animal Shelter: An animal shelter interface that the general public has access to view which displays the animals available for adoption. Volunteers are in charge of adding new animals as they show up to the shelter. Any user can put an available animal on hold, and adopt an animal that is on hold.

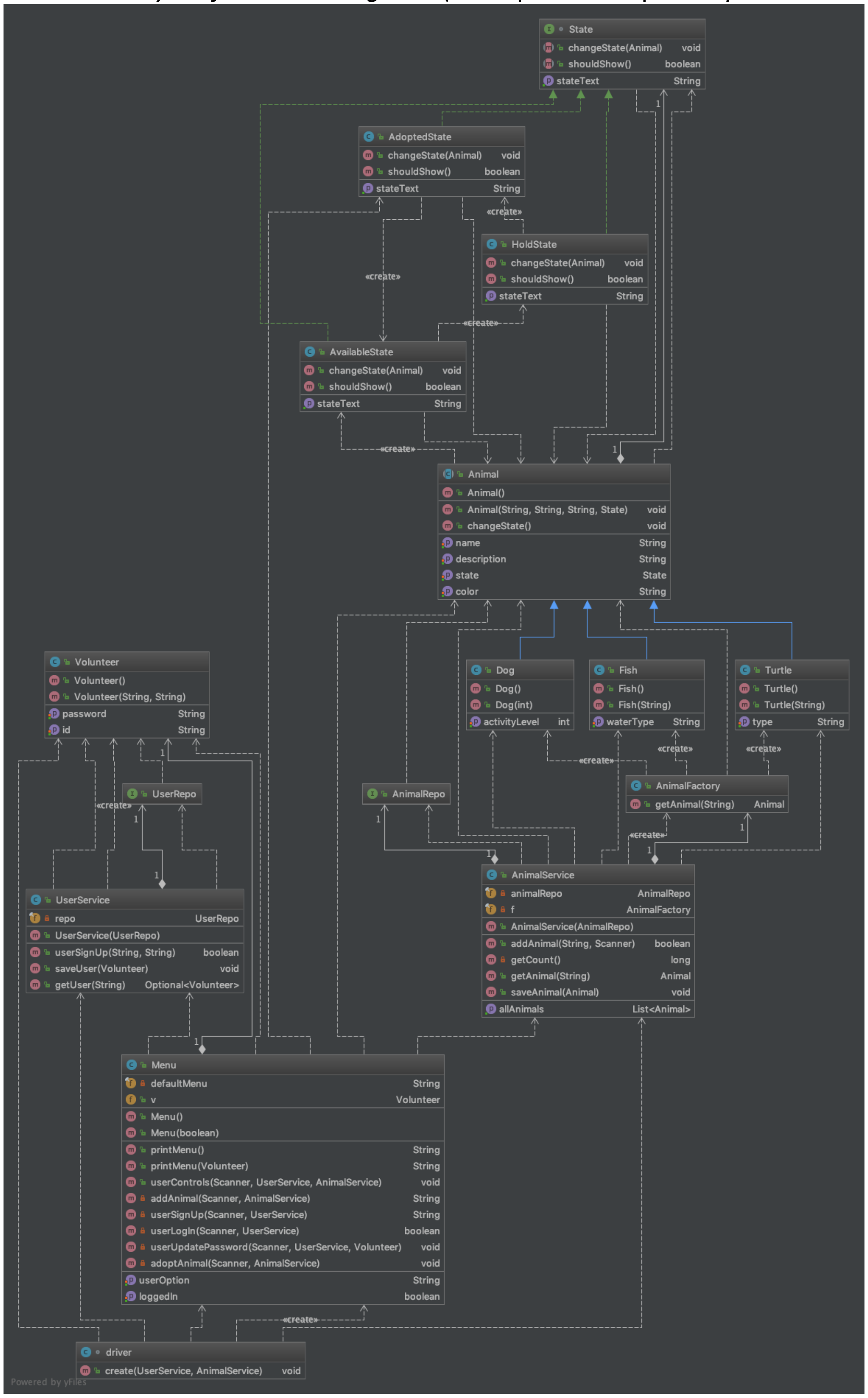
3. *List the features that were implemented (table with ID and title):*

ID	**User Requirement**
UR1	Customers can view animals of all type
UR2	Volunteers can log in
UR3	Volunteers can change their passwords
UR4	Volunteers can add animals
UR5	Customers can browse animals
UR7	Volunteers can log out
UR10	Customers can adopt animals

4. *List the features were not implemented (table with ID and title):*

ID	**User Requirement**
UR6	Customers can search for an animal by type
UR8	Volunteers can edit an animal's information
UR9	Volunteers can sign up using their employee id

5. *Show your final class diagram:* (also uploaded separately to the Github)



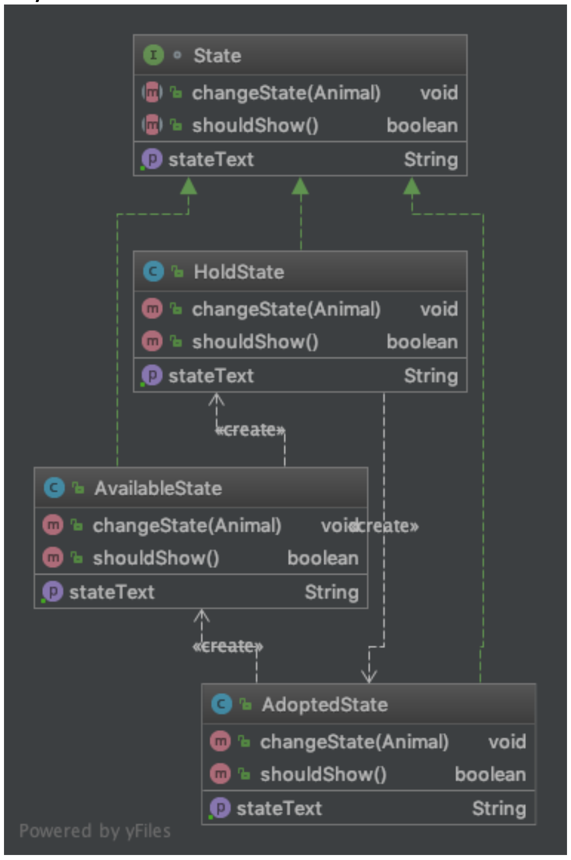
What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

The class diagram became much more complex with many classes added. Some classes such as driver and Menu were required in order to make the program run. Other classes such as UserService and AnimalService were important for creating and connecting to the database. The AnimalFactory was an attempt at using the factory design pattern to create animal objects, but it was not a true factory. The State interface, and the three classes that inherit from State, are implementing the State design pattern, referring to the state of the animal (whether it is available for adoption, on hold, or adopted).

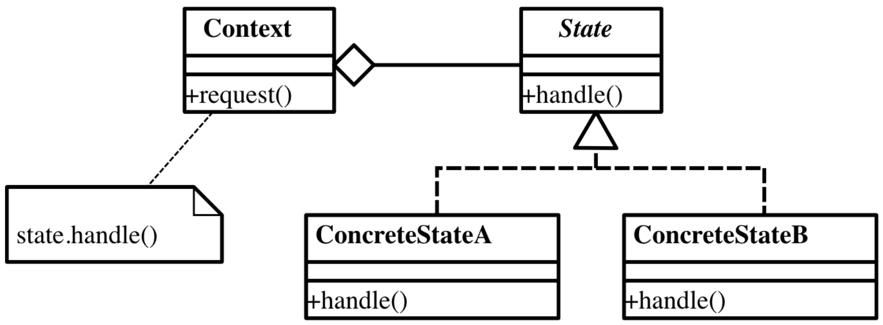
6. *For each design pattern implemented,*
    - *Show the classes from your class diagram that implement each design pattern.*
    - *Show the class diagram for the design pattern.*
    - *Explain how you implemented the design pattern, why you selected that design pattern.*

The State design pattern:

My classes



Generic class diagram



This design pattern was implemented by creating a State interface that had the functions necessary for each state change. Each class that implemented State had its own version of changeState(), shouldShow(), and stateText depending on whether the state was Available, On Hold, or Adopted. The AvailableState class implemented a changeState method that would change the state from Available to On Hold, a shouldShow method that returned true because an available animal should show up on the list of animals, and a stateText string that was set to “Available”. The HoldState class implemented a changeState method that would change the state from On Hold to Adopted, a shouldShow method that returned true because an animal on hold still shows up on the list of animals, and a stateText string that was set to “On Hold”. The AdoptedState class implemented a changeState method that would change the state from Adopted to Available (even though in the code this is not an option), a shouldShow method that returned false because an adopted animal should not show up on the menu, and a stateText string that was set to “Adopted”.

The design pattern was selected because it is a good design pattern for changing how the Animal is shown in the application without changing the class of the Animal, especially because there are different subclasses of animals. Without the design pattern, there may have had to be an AdoptedTurtle, AdoptedDog, or AdoptedFish class.

7. *What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?*

One thing I learned is that what you think will be a simple application can quickly become complex. My initial vision for this application had only 5 classes. Once I began coding the application, it became clear that I would not be able to get all of my functionality into 5 classes, and the number of classes rapidly grew from there. Creating an extremely simple but fully functional application in Java is much more complicated than I had originally expected.

Another thing I learned is that design patterns can clean up code that would otherwise be messy. My State design pattern saved me from creating new classes for each animal in each state (available, on hold, adopted). What could've been 9 classes ended up being only 6, and it was much cleaner and easier to understand and implement in the long run.