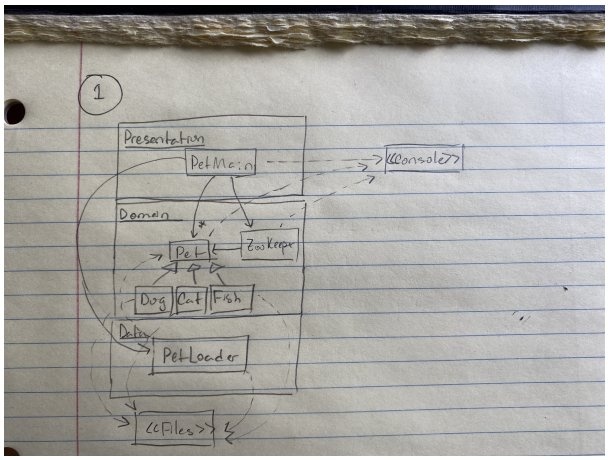
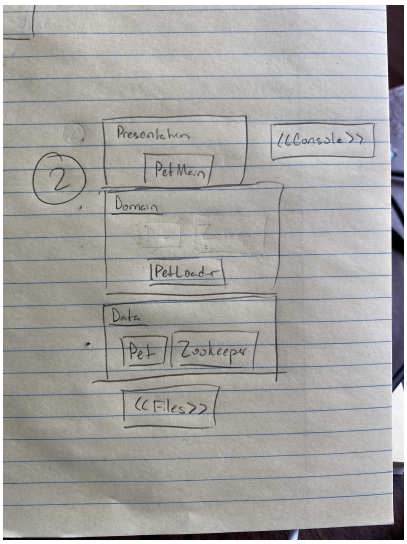
PetFactory Engineering Notebook

Entry 1: Rearranging Layers Based on Names

- Date: 3/26/2020
- Design Problem: The presentation, domain, and data source logic are all mixed up in the initial design.
- Candidate Designs:



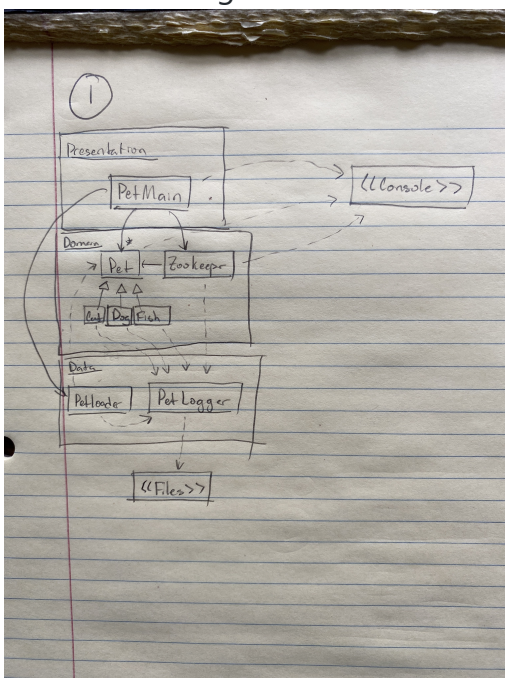


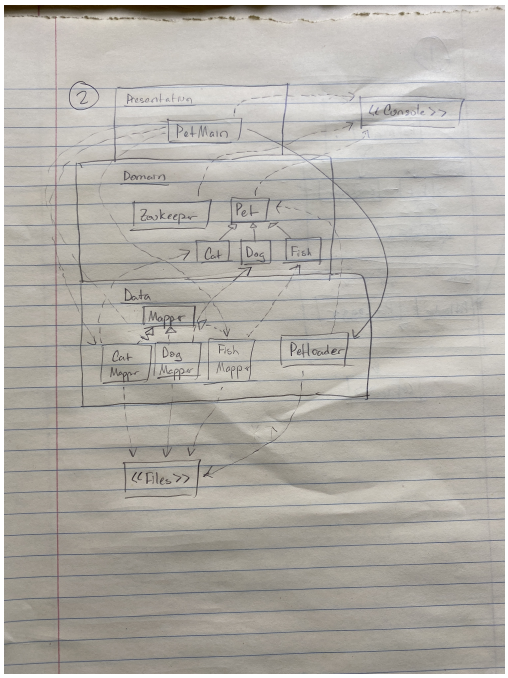
- Ultimate Decision and Rationale:
  - Favored Design
    - Design 1 mainly because the problem specified that only the name should be considered when separating the UML Classes into the three layers. All domain related objects should reside in the domain layers.
  - Discarded Design(s)

    - Design 2 was discarded because although the animals contain data, 1) Pets and Zookeeper themselves have intelligent behavior and 2) Pets and Zookeeper are part of the problem domain. The domain layers seems to be the better fit for Pets and Zookeeper
- Time Spent: 1.5 Hours

Entry 2: Adding a Data Layer Abstraction for Log Files

- Date: 3/27/2020
- Design Problem: The various pets all contain a layer violation due to the special abilities of all Pets causing a message to be written to a file. Find away to remove all dependencies that Pets have on Files.
- Candidate Designs:



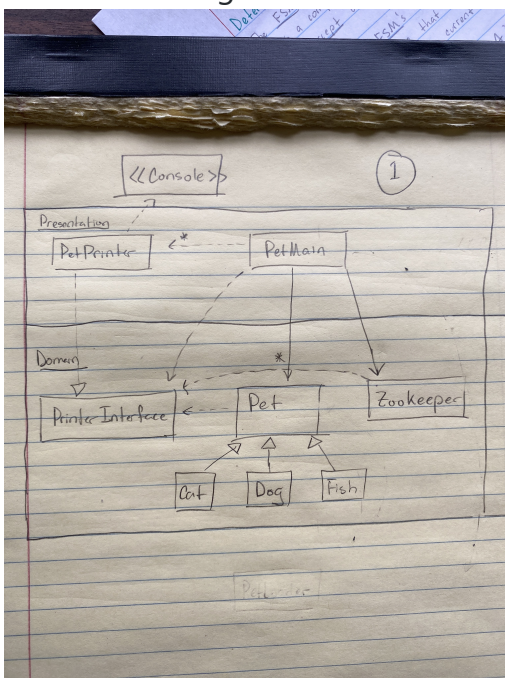


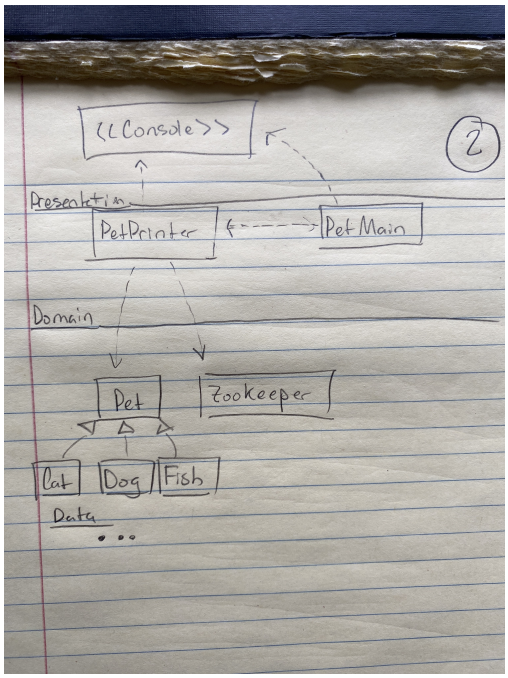
  - Notes: Candidate 1 Uses a Logger while Candidate 2 Uses a Data Mapper for every class.
- Ultimate Design Decision
    - Favored Design
        - Candidate Design 1 was the best candidate because it provides a simple interface for a File logging.
    - Discarded Designs
        - Candidate Design 2 would be better if interfacing with the database was more complex. Since the only interaction is logging messages to a File, it seems unneccesary to add more classes and dependencies to log files. This addition would be accounting for complexity that doesn't exist.

- Time Spent: 2 Hours

Entry 3: Moving Printing Logic Up to Presentation

- Date: 3/27/2020
- Design Problem: There is another layering violation. In the existing design, Pet is printing a message to the console. However, the printing logic has to exist in the Presentation Layer.
- Candidate Designs:



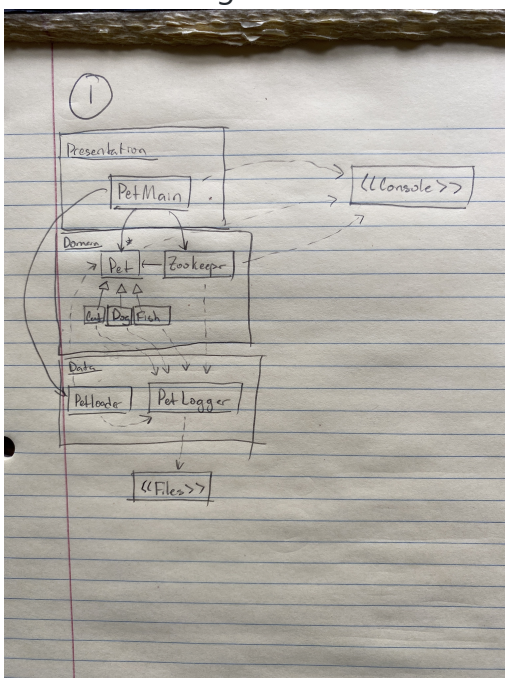


- Notes: Candidate 1 uses a Pet Printing interface that gets passed into Pet and Zookeeper. Candidate 2 uses a single Pet Printing Class that prints a message prepared by either Pet and Zookeeper. Several nonrelevant relations and classes were ommitted.

- Ultimate Design Decision
  - Favored Design
    - Candidate 1 because it provides an abstraction of message printing. Using this design, Pet or Zookeper donot need to know about how the messages are being displayed. The logic resides in PetPrinter.
  - Discarded Designs
    - Candidate 2 has Pet and Zookeeper prepare a message to print in bulk. This design forces console printing to be atomic, which solves a problem that's not in the original domain model. Candidate 1 is more relevant and loyal to the original design's functional intentions.
- Time Spent: 1 Hours

Entry 4: Change Zookeeper to Use the Data Layer

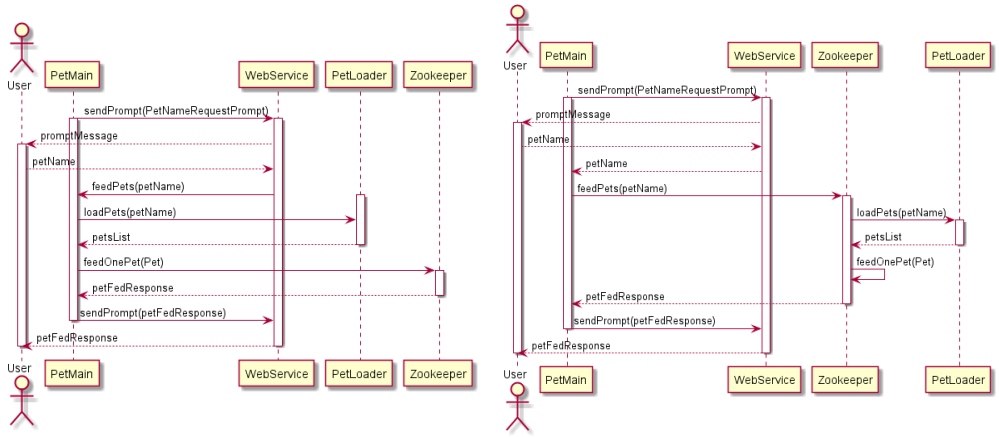
- Date: 3/27/2020
- Design Problem: When a zookeeper feeds a Pet, it logs it to Files. How can we reuse our previous design to resolve this violation?
- Candidate Designs:



- Notes: The logger dependency is applied to Zookeeper as well as Pet
- Ultimate Design Decision
  - Favored Design
    - Design 1: There aren't any other choices, mainly because there's no other reasonable way to reuse the previous design.
- Time Spent: 0.5 Hours

Entry 5: Change the user experience to be Web-compatible

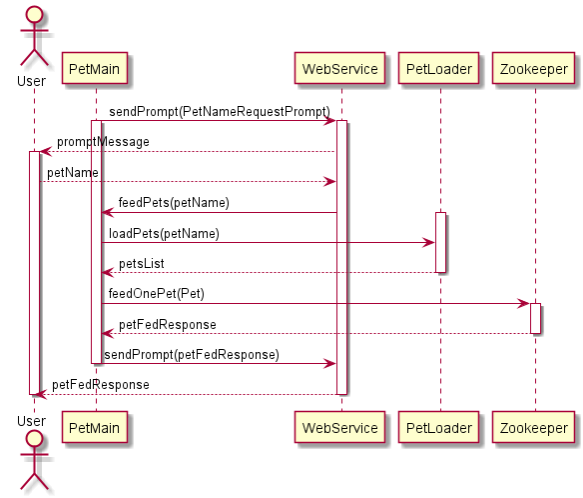
- Date: 3/27/2020
- Design Problem: Feedpets involves an infinite loop, which is not web compatibile
- Candidate Designs:

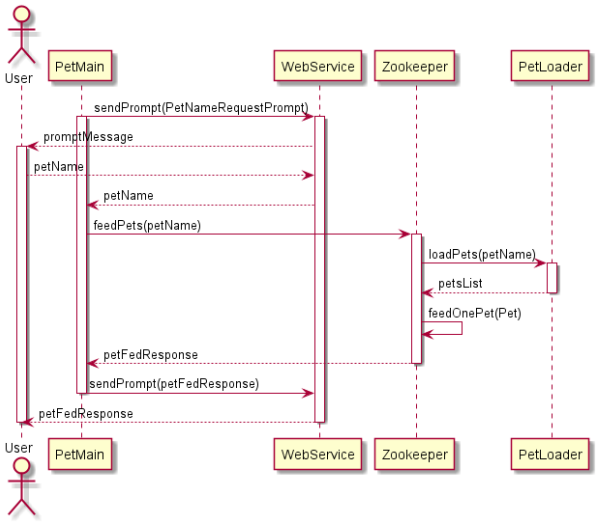


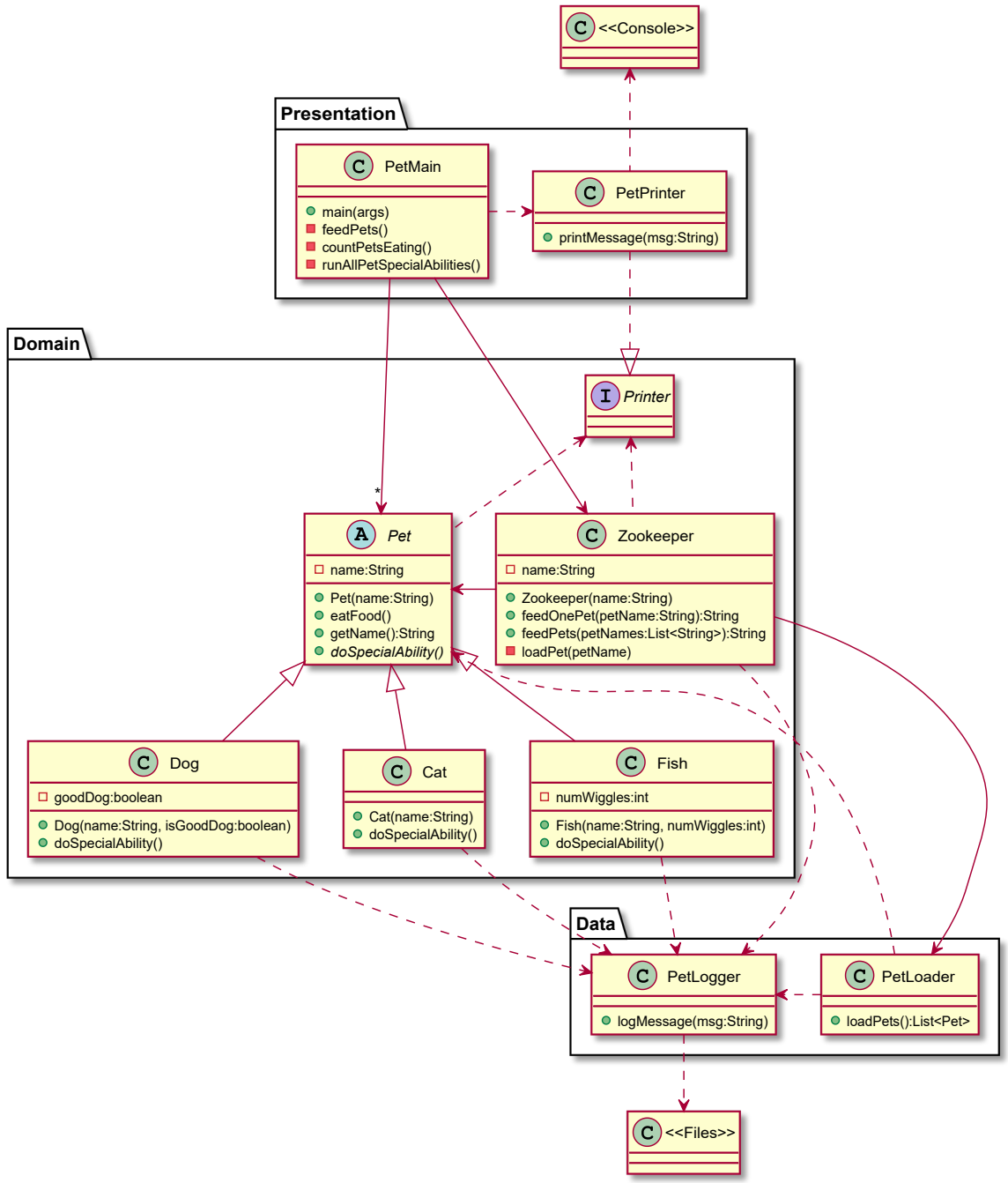
- Notes: Design 1 involves changing feedPets to include one petName string, finds the pet, and tells the zookeeper to feed it. Design 2 moves the feedPets function to the zookeeper and has the zookeeper handle the pet finding and feeding.

- Ultimate Design Decision
  - Favored Design
    - Design 2 because it composes all of the feeding logic to zookeeper, satisfying the "tell don't ask principle". PetMain shouldn't be worried about feeding pets.
  - Discarded Designs
    - Design 1. PetMain shouldn't have the responsibility of feeding pets. By moving feedPets to Zookeeper, functionality is more efficiently distributed.
- Time Spent: 1 hour

Final Class Diagram and Sequence Diagrams


User
PetMain
WebService
PetLoader
Zookeeper
sendPrompt(PetNameRequestPrompt)
promptMessage
petName
feedPets(petName)
loadPets(petName)
petsList
feedOnePet(Pet)
petFedResponse
sendPrompt(petFedResponse)
petFedResponse
User
PetMain
WebService
PetLoader
Zookeeper


User
PetMain
WebService
Zookeeper
PetLoader
sendPrompt(PetNameRequestPrompt)
promptMessage
petName
petName
feedPets(petName)
loadPets(petName)
petsList
feedOnePet(Pet)
petFedResponse
sendPrompt(petFedResponse)
petFedResponse
User
PetMain
WebService
Zookeeper
PetLoader


<<Console>>
Presentation
PetMain
main(args)
feedPets()
countPetsEating()
runAllPetSpecialAbilities()
PetPrinter
printMessage(msg:String)
Domain
Printer
*
Pet
name:String
Pet(name:String)
eatFood()
getName():String
doSpecialAbility()
Zookeeper
name:String
Zookeeper(name:String)
feedOnePet(petName:String):String
feedPets(petNames:List<String>):String
loadPet(petName)
Dog
goodDog:boolean
Dog(name:String, isGoodDog:boolean)
doSpecialAbility()
Cat
Cat(name:String)
doSpecialAbility()
Fish
numWiggles:int
Fish(name:String, numWiggles:int)
doSpecialAbility()
Data
PetLogger
logMessage(msg:String)
PetLoader
loadPets():List<Pet>
<<Files>>