This assignment is intended to be done by your team of two students. You may collaborate on answers to all questions or divide the work for the team. In any case, the team should review the submission as a team before it is turned in.

Project 4 is intended as a continuation of Project 2 and 3's simulation of the Friendly Neighborhood Pet Store (FNPS). You may reuse code and documentation elements from your earlier project submissions. You may also use example code from class examples related to Project 2. In any case, you need to cite (in code comments at least) any code that was not originally developed by your team.

Part 1: UML Exercises and Semester Project Proposal – 25 points

1) (5 points) Provide a one-page project proposal for Projects 5/6/7 (aka the Semester Project). A typical project will involve development of a user interface, a data source, and internal program logic for operations. Examples of past projects are on Canvas here: https://canvas.colorado.edu/courses/83585/discussion_topics/915379
   Your program can be a web or mobile app, a game, a utility, a simulation – really anything that can be demonstrated for its operation. It must be in an Object-Oriented language (sorry C folks) as you will be required to demonstrate OO patterns, but the language does not have to be Java. Generally, for scoping the size of the program, each team member should plan to implement two to four use cases or functional program elements in projects 6 and 7. Project 5 will be a design effort with required deliverables to be detailed later. Your proposal should include:
   a. Title
   b. Team Members
   c. Description paragraph
   d. Language choice (including any known libraries or frameworks to be used)
   e. List of 2 to 4 functional elements per team member (ex. Login screen, Game piece graphics, etc.)

   In addition to the project proposal submission, please add an entry to this Google Doc to reserve your project: https://docs.google.com/document/d/1avbEnD8BJX3XA5MwtaZo-7EyFAQ4F3oR8t5ar6OPArk/edit?usp=sharing

2) (10 points) UML Sequence Diagram for Project 3. Select at least four top level active objects for your simulation (Ex. Clerk, Trainer, Store, System.out, etc.). Create a sequence diagram that shows the primary message or method invocations between these objects for the tasks performed by Employees in a day. The sequence diagram can be just the happy path, it does not have to show error conditions. The diagram should show object lifetimes during the operations.

3) (10 points) UML Class Diagram for Project 4 Part 2. Draw a class diagram extending the FNPS simulation as described in Project 4 Part 2. The class diagram should contain any classes, abstract classes, or interfaces you plan to implement. Classes should include any key methods or attributes (not including constructors). Delegation or inheritance links should be clear. Multiplicity and accessibility tags are optional. You should note what parts of your class diagrams are implementing the three required patterns below: Factory, Singleton, and Command

Part 2: FNPS simulation extended – 50 points (with possible 10 point bonus)

Using the Project 2 and 3 Java code developed previously as a starting point, your team will create an updated Java program to simulate extended daily operations of the FNPS. The simulation should perform all functions previously enabled in Project 3. Employees will continue to perform all functions they performed in Project 3. The simulation will be refactored with the following extensions.

1) The simulation will be run each day for two stores (Northside FNPS and Southside FNPS). The stores will operate independently with their own cash flow and inventories.
2) A pool of employees will serve both stores. An employee may work at one store on day 1, and the other on day 2. Determine the minimum number of employees needed to keep both stores open given the limits on the number of days in a row an employee can work AND the possibility of sick employees each day that will not be available to work.
3) Implement a command line interface to allow a user to act as a customer. You will run the stores for 10 to 30 days (randomly or entered as a prompt, you can choose). At the end of the 10 to 30 days, the next day will start as normal until the opening of the store. At that point, you will present a command line interface to allow a user to interact with the stores (there will be no random customers for this day).
   The command line interface should allow the user to issue commands to a clerk and should be modeled using a **Command pattern**. Commands will include:
    a. Select a store to issue commands to (this can be done at any time)
    b. Ask the employee their name (should reply with employee's name)
    c. Ask the clerk what time it is (should return the system time)
    d. Ask the trainer for current store inventory (to allow selecting an item)
    e. Ask the trainer for information on a user selected inventory item
    f. Buy a normal inventory item from the clerk (if the item is in inventory), again, this should follow the normal purchase flow – the user can decide not to buy, the trainer will offer a discount, etc.
    g. End the user interactions

   When the interactions end, the employees should go through the normal remaining store actions, and the simulation should end with appropriate summary reports as usual.
4) Implement **Factory patterns** to create inventory items and employees, separating creation logic from use logic for these elements.
5) Modify the Logger and Tracker objects to be created from **Singleton patterns**, there must only be one of each observer object made – both stores will send their events to the objects, and the objects should show information for each store separately – this means a logger-n.txt file with events by store and Tracker summaries organized by store. You are required to code one Singleton using **lazy instantiation**, the other should use **eager instantiation**.
6) Include JUnit (version 4 or 5) tests in your code (minimum of 15 tests). These tests can verify objects are instantiated, check expected numeric values, or test algorithms in the code. You must be able to run your code in test mode to run these JUnit tests and capture the output of the tests to a test output text file in your repository. See JUnit references in the prior Project 3 instructions. Note that if you created JUnit tests that you bring over from the Project 3 bonus exercise, you only need to add an additional 5 tests.

Simulate the running of the FNPS store for 30 days. At the end of the 30 days, print out a summary of the state of the simulated store as before, including:

- the items left in inventory and their total value (purchasePrice)
- the items sold, including the daySold and the salePrice, with a total of the salePrice
- the pets remaining in the sick pets collection
- the final count of money in the Cash Register
- how much money was added to the register from the GoToBank Action

There may be possible error conditions due to insufficient funds or lack of inventory that you may need to define policies for and then check for their occurrence. You may find requirements are for this simulation are not complete in all cases. In such cases, either document your interpretation or find class staff for clarifications.

The code should be in Java 8 or higher and should be object-oriented in design and implementation.

In commenting the code, clearly indicate where instances of the Factory, Singleton, and Command patterns are used. Also highlight the eager or lazy instantiation within your Singletons.

Capture all announcement output from a single simulation run in your repository in a text file called Output.txt (by writing directly to it or by cutting/pasting from a console run). This should now include output from the Tracker object as well. You should also include each of the "Logger-n.txt" files created by the Logger object in your repo.

Bonus Work – 10 points for two Line Charts

There is a 10-point extra credit element available for this assignment. For extra credit, import a charting library to create two line graphs. One graph should show two lines – Item Sales and Total Register $ – for each day of the simulation. Another graph should show three lines – counts of Items in Inventory, Sick Pets, and Items Sold – for each day of the simulation. Java charting libraries in common use include: JFreeChart (https://www.jfree.org/jfreechart/), charts4j (https://github.com/julienchastang/charts4j), and XChart (https://github.com/knowm/XChart). To receive all bonus points, graph generation code must be part of your code and clearly commented, and the output for the graphs must be captured as images in a PDF or other viewable files included in your repo.

Grading Rubric:

Homework/Project 3 is worth 75 points total (with a potential 10 bonus points in part 2). The project is due on Canvas as a URL to your repository on Wednesday 6/29 at 8 PM.

Part 1 is worth 25 points. The diagrams for Part 1 should be captured in a PDF that should also contain the names of all team members. The PDF should be included in the project repository.

Question 1 should provide all requested information about your proposed Semester Project work (-1 for missing items, -5 for a missing proposal).

Questions 2 and 3 should provide thorough UML diagrams that could be followed to produce or understand the FNPS simulation program in Java with the new changes and patterns. This includes identifying major contributing or communicating classes/objects (ex. Items, Employees, etc.) and methods or attributes found in their design. As stated, multiplicity and accessibility tags in class diagrams are optional. Use any method reviewed in class to create the diagram **that provides a readable result**, including diagrams from graphics tools or hand drawn images. **The elements of the diagram that implement the Factory, Singleton, and Command**

patterns should be clearly annotated in the class diagram. A considered, complete UML diagram will earn full points, poorly defined or clearly missing elements will cost -1 to -2 points, missing the diagram is -10 points.

Part 2 is worth 50 points (plus possible 10 point bonus). The submission will be a URL to a GitHub repository. The repository should contain well-structured OO Java code for the simulation, a captured Output.txt text file with program results, the Logger-n.txt files, the PDF of Part 1, and a README file that has the names of the team members, the Java version, and any other comments on the work – including any assumptions or interpretations of the problem. The bonus graph captures should also be in the repository if created. Only one URL submission is required per team.

25 points for comments and readable OO style code: Code should be commented appropriately, including citations (URLs) of any code taken from external sources. We will also be looking for clearly indicated comments for the three patterns to be illustrated in the code. A penalty of -2 to -4 will be applied for instances of poor or missing comments, poor coding practices (e.g. duplicated code), or excessive procedural style code (for instance, executing significant program logic in main).

20 points for correctly structured output as evidence of correct execution: The output from a run captured in the text file mentioned per exercise should be present, as should be the set of Logger-n.txt files. A penalty of -1 to -3 will be applied per exercise for incomplete or missing output.

5 points for the README file: A README file with names of the team members, the Java version, and any other comments, assumptions, or issues about your implementation should be present in the GitHub repo. Incomplete/missing READMEs will be penalized -2 to -5 points.

10 bonus points for including both graph generation in your code (clear and commented), and the output for the graphs captured as images in a PDF or other viewable files included in your repo.

Please ensure all class staff are added as project collaborators (brmcu, gayathripadmanabhan1, wmaxdonovan-cu) to allow access to your private GitHub repository. Do not use public repositories.

Overall Project Guidelines

Be aware that the class midterm exam will run from noon Saturday 6/25 through the end of the day Thursday 6/30. The exam window will be three hours (longer for those with accommodation). Plan ahead to make sure you allocate time for this project work and the midterm. To help with this, I will waive the normal 5% late penalties for the first two days after the due date, but the 15% penalty for the last two days will still be in place.

Assignments will be accepted late for four days. There is no late penalty within 4 hours of the due date/time. In the next 48 hours, for this assignment only, the 5% late penalty will be waived. Assignments submitted in the next 48 hours after that will have the standard late penalty of 15% of the grade. After this point, assignments will not be accepted.

Use e-mail or Piazza to reach the class staff regarding homework/project questions, or if you have issues in completing the assignment for any reason.