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 the final evolution of the **Friendly Neighborhood Car Dealer (FNCD)** simulation. You may reuse code and documentation elements from your Project 2 and 3 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

Provide answers to each of the following in a PDF document:

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/91069/discussion_topics/1075071

   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.)

   Note that games of chess, Monopoly, Battleship, and Catan are not allowed (too many past instances).

   In addition to the project proposal submission, please add an entry to this Google Doc to reserve your project (try not to directly duplicate another team's work): https://docs.google.com/document/d/1t3aYEj97PPZtIUL9PuyRNSXZDo_hrrHPuT4GH3F97gU/edit?usp=sharing

2) (10 points) UML Sequence Diagram for Project 4. Select at least four top level active objects for your simulation (Ex. FNCD, Vehicle, Staff, etc.). Create a UML Sequence Diagram that shows the primary message or method invocations between these objects for the tasks performed in a typical simulation 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 (if they change or not).

3) (10 points) Draw a class diagram for extending the FNCD simulation 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: FNCD 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 extending functionality of the FNCD simulation. The simulation should generally perform as previously enabled in Project 2 and 3 allowing for the simulation code changes to be refactored as follows:

Change Summary

- One additional FNCD
- Three additional Vehicles and Inventory Change
- Change to a human-controlled Interface to simulate a Buyer
- Command Pattern for interacting with the user and the simulation functions
- Singleton Pattern for selected objects (using lazy and eager instantiation)
- Factory Pattern for creation of Vehicles and Staff
- 15 required JUnit test assertions

One additional FNCD

A second FNCD is opening! Run two FNCDs in parallel in some fashion. At a minimum, in a single threaded approach, a daily activity should complete at all FNCDs before going on to the next daily activity. The FNCDs are named North and South.

Three additional Vehicles and Inventory Change

Create three new subclasses of vehicles. You may select all the characteristics of these vehicles, including their types and any new attributes or methods. You will determine whether vehicles will participate in races.

The Inventory count of all Vehicle type instances at an FNCD changes from 4 to 6. At least six of each vehicle type should be created at startup, and each day, additional vehicles should be added to keep to six at the start of any given day.

A Human-controlled Interface to allow a user to act as a Buyer

Implement a command line interface to allow a user to act as a customer. You will run the stores for 30 days as usual. At the end of the 30 days, day 31 will start as normal until the beginning of sales at each FNCD. At that point, you will present a command line interface to allow a user to interact with the FNCD of their choosing (there will be no random customers for this day). After the user ends the interaction, close out day 31 as usual.

The command line interface should allow the user to issue one of eight commands to a salesperson (selected randomly at each location) and should be modeled using a Command pattern. You can select the method used for interaction – a numeric menu, string command entry, etc.

Commands will include:

- Select one of the FNCD locations to issue commands to (this can be done at any time)
- Ask the salesperson their name (should reply with employee's name)
- Ask the salesperson what time it is (should return the system time)
- Ask for a different salesperson (select another salesperson for the transactions)
- Ask the salesperson for current store inventory (to allow selecting an item)

- Ask the salesperson for all details on a user selected inventory item
- Buy a normal inventory item from the salesperson (if the item is in inventory), again, this should follow the normal purchase flow – the user may decide to buy or not buy, if they do buy, they may choose to buy or not buy add-ons.
- End the user interactions

Singleton Pattern for selected objects (using lazy and eager instantiation)

You must create the Logger and Tracker objects as full Singleton implementations, creating one using a lazy instantiation, and one with an eager instantiation at your choice. Point out the lazy/eager method with clear comments. Logger and Tracker should function using the Observer pattern largely as implemented in Project 3, creating Logger-n.txt files, etc. You may use Singleton elsewhere as well if appropriate. Only one Logger and Tracker should be used for all active FNCD output.

Factory Patterns for creation of Vehicles and Staff

Your Vehicle and Staff creation should be encapsulated in creation focused Factory pattern code. You should use Enums to clearly request different types of Vehicles and Staff as products of subclassed concrete creators. You may use Abstract Factory if you feel it better fits your particular implementation.

15 required JUnit test assertions

You must implement 15 JUnit assertions as described in the bonus work for Project 3. You must capture the results of running the tests with a screen print or other output and include it clearly in your repo. See Project 3 for support links for adding JUnit assertions to your project.

Outputs, Comments, UML Class Diagram Update

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

Captured output: Run the simulation for 30 simulated days plus the user interaction. All noteworthy events or state changes in the simulation should generate output as in Project 3. New events should generate new messages. Capture the output from the entire run in a text file called SimResults.txt. Ensure that you capture examples of all interactive user actions from day 31. Also capture output from Logger and Tracker as normal.

Identify OO Patterns: In commenting the code, clearly identify your implementations of Factory, Singleton, and Command. Note that you can use Factory, Singleton, and Command in more than just the functionality outlined if you wish.

Include a UML class diagram update: Also include in your repository an updated version of the FNCD UML class diagram from 4.1 that shows your actual class implementations in project 4.2. Note what changed between part 4.1 and part 4.2 (if anything) in a comment paragraph.

Errors, boundary conditions, incomplete specifications: There may be possible error conditions that you may need to define policies for and then check for their occurrence. These include running out of Vehicles to sell or money in the FNCD budgets. You may also find requirements are not complete in all cases. Document any assumptions you make in the project's README file. See class staff for any guidance needed.

Bonus Work – 10 points for a Java-based Line Graph

There is a 10-point extra credit element available for this assignment. For extra credit, import a charting library to create a line graph of simulation events. The graph should be generated at the end of the simulation run. The X axis should be the days in the simulation, from 1 to 30 (or 32, your choice). The Y axis should be count and dollar values. The lines should represent number of Vehicles sold, money earned by Staff, and money earned by FNCD. Create one graph for each FNCD instance.

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 clear and commented, and the output for the graphs must be captured as images in a PDF or other easily viewable files included in your repo.

Grading Rubric:

Homework/Project 4 is worth 75 points total (with a potential 10 bonus points for part 2)

Part 1 is worth 25 points and is due on Wednesday 3/8 at 8 PM. The submission will be a single PDF per team. The PDF must contain the names of all team members.

Question 1 will be scored on completeness of your proposal (-1 per missing items). You will be contacted if there are concerns or questions about your proposal.

Question 2 will be scored based on your effort to provide a thorough UML sequence diagram that shows the flow of your Project 3 simulation. Poorly defined or clearly missing elements will cost -1 to -2 points, missing the diagram is -10 points.

Question 3 should provide a UML class diagram that could be followed to produce the FNCD simulation program in Java with the new changes and patterns. This includes identifying major contributing or communicating classes and any methods or attributes found in their design. As stated, multiplicity and accessibility tags 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 Command, Singleton, and Factory patterns should be clearly annotated.** 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) and is due Wednesday 3/15 at 8 PM. The submission will be a URL to a GitHub repository. The repository should contain well-structured OO Java code for the simulation, SimResults.txt, the Logger-n.txt files, proof of JUnit run in a file labeled JUnitResults of appropriate readable file type, the updated UML class diagram for Part 2, the bonus Line Graph images (if provided) 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. Only one URL submission is required per team.

20 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).

15 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 points for the updated UML file showing changes from part 1 to part 2 as described. Incomplete or missing elements in the UML diagram will be penalized -2 to -4 points.

10 point possible bonus for clearly identified graph generation code AND a line graph generated by the Java code included in the repository in an easily readable format.

Please ensure all class staff are added as project collaborators to allow access to your private GitHub repository. Do not use public repositories.

Overall Project Guidelines

Be aware that the class midterm will run from noon Saturday 3/11 through the end of the day Thursday 3/16. The exam window will be three hours (longer for those with accommodation). Plan ahead to make sure you allocate time for the project work and the midterm. To help with this, I will waive the 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. Please do not ask for extensions without severe need.

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, the penalty for a late submission is normally 5%, but is waived for this assignment. In the next 48 hours, the late penalty increases to 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.