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 submissions as a team before they are turned in.

Project 3 is intended as a continuation of the Project 2 simulation of the **FNCD (Friendly Neighborhood Car Dealership)**. You may reuse code and documentation elements from your Project 2 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 – 20 points

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

1) (10 points) Considering the events and flow of the extended FNCD simulation described in Project 3 Part 2, create either a detailed UML Activity diagram to describe the flow of actions and decision points in the simulation, or a detailed UML State diagram that shows program states and transitions that cause state changes. Which one you use likely depends on whether your code is more code flow or state/transition based.
2) (10 points) Draw a class diagram for extending the FNCD simulation as it is described in Project 3 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: Strategy, Decorator, and Observer.

Part 2: FNCD simulation extended – 55 points (with possible 10 point bonus)

Using the Project 2 Java code developed previously as a starting point, your team will create an updated Java program extending the elements of the FNCD simulation. The simulation should perform all functions previously enabled in Project 2. Generally, Staff and Vehicles will be used to perform all functions they performed in Project 2. The simulation code will be refactored as follows:

Change Summary

- Additional Vehicle and Staff Types
- The FNCD is now open every day – for racing! – Sundays (when previously the FNCD was closed) and Wednesdays are now Race Days, with a Race activity occurring just before the Ending activity that day
- Strategy Pattern added for assigning Wash methods to Interns
- Decorator Pattern used for add-on purchases for Vehicles being sold to buyers
- Observer Pattern added with Logger and Tracker subscribers

Additional Vehicle and Staff Types

The FNCD is adding several types of new classes of used Vehicles: Electric Cars, Motorcycles, and Monster Trucks. Each new subclass of Vehicle will be created as other vehicles with a cost from a cost range, an initial price (2x cost), a Cleanliness, and a Condition.

Electric Cars have a unique Range attribute, initially set randomly from 60 to 400 miles. If an Electric Car arrives as Like New or is repaired to Like New, Range increases by 100 miles.

Motorcycles have an unique Engine Size rating in cubic centimeters (cc). This value should be generated from a truncated Normal Distribution with mean 700 std. dev. 300. In no case can cc be less than 50.

Monster Trucks must have a unique Stage Name associated with famous Monster Trucks as listed in the https://www.rookieroad.com/monster-trucks/list-a-z-2027269/ website. If you are forced to reuse a name, it should gain a sequential numeric ending: e.g. Bigfoot 2 or Zombie 6.

A new type of staff member, Driver, will be added. As with other Staff, three Drivers are active at all times. Drivers do not quit like other Staff subclasses. However, they may be Injured in a Race activity. In that case, they leave the FNCD and are replaced by a new Driver the following day.

Race Events (Sunday and Wednesday)

Randomly select a type of vehicle to send to the races (regular Cars and Electric Cars do not race, all other types do). Three of the Vehicles in the current inventory of the selected type will be sent to the races to participate as long as they are not Broken (If there are not enough running Vehicles or vehicles of that type, the number sent to the race may be reduced, or the FNCD may be unable to participate at all.) Each vehicle sent will be associated with one of the FNCD drivers for that race.

A total of 20 Vehicles (up to 3 of which are from the FNCD) will participate in each Race event. For the race, randomly determine a final race position for each FNCD Vehicle from 1 to 20. The first three race positions (1, 2, and 3) are considered Winners. Slotting into one of the last five race positions will label the Vehicles as Damaged.

If one of the FNCD Vehicles is a Winner, it will increment a count of races won which will be kept as an attribute of that Vehicle instance. Vehicles that have a Win count of 1 or more increase their sale price in the Selling Activity by 10%. Likewise, the Driver of the Vehicle will increment a count of races won, kept with the Driver instance. The Driver receives a cash bonus if they drove a Winner Vehicle.

If the Vehicle is in the Damaged category, the Vehicle's condition goes to Broken, and the Driver of the Vehicle has a 30% chance of being Injured.

All race participants and outcomes should be announced in simulation output.

Strategy Pattern for Intern Wash behavior

Each Intern will be assigned a Washing Method when instantiated – Chemical, Elbow Grease, or Detailed. (The results of each type of Washing Method will replace the previous Wash outcomes from Project 2.) Interns continue to receive bonuses for Sparkling outcomes. Possible Washing Methods:

- Chemical:
    - Dirty Vehicle: 80% chance of Clean, 10% chance of Sparkling
    - Clean Vehicle: 10% chance of Dirty, 20% chance of Sparkling
    - In all cases, a 10% chance of Vehicle becoming Broken (if not already)

- Elbow Grease:
    - Dirty Vehicle: 70% chance of Clean, 5% chance of Sparkling
    - Clean Vehicle: 15% chance of Dirty, 15% chance of Sparkling
    - In all cases, 10% chance of Vehicle becoming Like New (if not already)
- Detailed:
    - Dirty Vehicle: 60% chance of Clean, 20% chance of Sparkling
    - Clean Vehicle: 5% chance of Dirty, 40% chance of Sparkling
    - No special effects as above methods

When an Intern washes a Vehicle, announce what Washing Method is in use.

Decorator Pattern for add-on purchases for Vehicles sold to Buyers

Buyers will be offered 4 add-on purchases after deciding to buy a Vehicle. These should be added as decorators for the Vehicle class with a name and an additional price. Buyers will be offered the following 4 add-ons:

- Extended Warranty = 20% of Vehicle sale price, 25% chance of Buyer adding
- Undercoating = 5% of price, 10% chance of adding
- Road Rescue Coverage = 2% of price, 5% chance of adding
- Satellite Radio = 5% of price, 40% chance of adding

When a Salesperson sells a Vehicle, announce any add-on purchases and price increases.

Observer Pattern with Logger and Tracker subscribers

Add an Observer pattern implementation to allow subscription for and publication of events. You may create a custom Observer framework, or you may use a library (e.g. Java Flow) or other publish/subscribe tool. You may use a pull or push model for events. Event messages can be text, JSON, or specialized event objects of your design.

Use the Observer implementation to create two new summaries of simulation events in two subscriber objects.

- Publish the following events out to subscribers:
    - Adding money to the FNCD budget due to low funds
    - Washing, Repair, and Sales outcomes
    - Race attendance and results
    - Any changes in money (salary, bonus, sales, etc.) for staff or the FNCD
- Sufficient information should be provided in the published event that the subscriber does not have to query other objects (not including the publisher) for additional information.
- Create an event consumer class called a Logger. The Logger object should be instantiated at the beginning of each full simulation day and should close at the end of each day. The Logger object should subscribe for the published events that occur during a simulated day and write each of them in a human readable form as they are received to a text file named “Logger-n.txt” where n is the day number of the simulation.
- Create an event consumer class called a Tracker. The Tracker object will be instantiated at the beginning of the simulation run and stay active until the end. The Tracker will subscribe for the published events and maintain a data structure in memory for tracking the total money earned by staff and by the FNCD since the first day. At the end of each day the Tracker should print a summary of the cumulative data like:

```
Tracker: Day 4
Total money earned by all Staff:  $27500.00
Total money earned by the FNCD:  $56900.00
```

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. All noteworthy events or state changes in the simulation should generate output as in Project 2. New events should generate new messages. Capture the output from the entire run in a text file called **SimResults.txt**.

Identify OO Patterns: In commenting the code, clearly identify your implementations of **Strategy, Observer**, and **Decorator**. Note that you can use Strategy, Observer, and Decorator 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 3.1 that shows your actual class implementations in project 3.2. Note what changed between part 3.1 and part 3.2 (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 budget. 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 JUnit test example

There is a 10-point extra credit element available for this assignment. For extra credit, import a version of JUnit of your choice, and use at least ten JUnit test (assert) statements to verify some of your starting expected objects are instantiated or to perform other similar functionality tests. For full bonus points you must document how you run your JUnit tests (e.g. with a command line or in the IDE), and you must capture output that shows the results of your tests. You can decide how the test methods are integrated with your production code.

In practice, writing your tests before development is recommended, but for this academic example, I recommend you do not pursue this bonus work until you are sure the simulation itself is working well. If you need support on using JUnit, I mention several references in the TDD lecture, but here are key helpful ones:

- The JUnit sites for JUnit 5 (https://junit.org/junit5/) and JUnit 4 (https://junit.org/junit4/)
- The Jenkov JUnit tutorials (they are for JUnit 4, but are extremely clear and helpful regardless): http://tutorials.jenkov.com/java-unit-testing/index.html
- Organizing your JUnit elements in your code: https://livebook.manning.com/book/junit-recipes/chapter-3/1

Grading Rubric:

Homework/Project 3 is worth 75 points total (with a potential 10 bonus points for part 2)

Part 1 is worth 20 points and is due on Wednesday 2/22 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 based on your effort to provide a thorough UML activity or state diagram that shows the flow of your Project 3 simulation. Poorly defined or clearly missing elements will cost -1 to -3 points, missing the diagram is -15 points.

Question 2 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 Observer, Strategy, and Decorator 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 55 points (plus possible 10 point bonus) and is due Wednesday 3/1 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, the updated UML class diagram for Part 2, 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).

20 points for correctly structured output as evidence of correct execution: The output from a run captured in the text file mentioned should be present, as should be the set of Logger-n.txt files. A penalty of -2 to -4 points will be applied for incomplete or missing output elements.

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.

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

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 5%. 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.