This assignment is intended to be done by your team of two to three 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 the Friendly Neighborhood Music Store (FNMS) simulation from Project 2 and 3. You may reuse code and documentation from your Project 2 or 3 submissions. You may also use example code provided in class. However, 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/79638/discussion_topics/843623
   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/1UmJ38QxcBRZKh3ZhUUxojkSlGwFYI47_n36K4MtY4ZQ/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, Store, System.out, etc.). Create a sequence diagram that shows the primary message or method invocations between these objects for the tasks performed by a Clerk 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 for extending the FNMS 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: Abstract Factory, Singleton, and Command

**Part 2: FNMS simulation second revision – 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 FNMS. The simulation should continue to perform all functions previously enabled in Project 3. Clerks 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 FNMS and Southside FNMS). The stores will operate independently with their own cash flow and inventories.
2) A pool of clerks will serve both stores. Determine the minimum number of clerks needed to keep both stores open six days a week with the possibility of one or two sick clerks 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 the 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 clerk their name (should reply with clerk's name)
   c. Ask the clerk what time it is (should return the time)
   d. Sell a normal inventory item to the clerk – following the normal selling to a store transaction flow – the user can decide whether to accept the buying price the clerk offers at the steps of the sale
   e. 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 clerk will offer a discount, etc.
   f. Buy a custom guitar kit from the clerk
   g. End the interactions

   When the interactions end, the clerk should go through the normal remaining store actions, and the simulation should end with appropriate summary reports as usual.
4) Only the interactive user will ever purchase a custom guitar kit. A custom guitar kit is made up of the following parts: (1) bridge, (2) knob set, (3) covers, (4) neck, (5) pickguard, and (6) pickups. You must allow the customer to choose from 1 of 3 choices in each category (the choices can be named or you can just use letters – pickups A, pickups B, pickups C for instance). Each individual part will have a price. Provide the overall kit and price of the kit as a final product to sell to the user. There is no inventory control for the parts – it is assumed you always have them. The custom guitar kit should be made from an **Abstract Factory pattern** designed specifically for either the south side or north side store, and each store will have different sets of parts used to produce a guitar kit. A guitar kit, once produced, should be treated as an item, and should be tracked as a sold item.
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 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 summary tracking by store – note: one singleton should be coded 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. One way to capture output summaries from test runs is to use the JUnit TestWatcher API detailed here (https://www.baeldung.com/junit-testwatcher), but any output that shows all test results is sufficient. See other JUnit references in the prior Project 3 description. If you created JUnit tests that you bring over from Project 3, you only need to add an additional 5 tests.

Simulate the running of the FNMS store for 10 to 30 days and also capture examples of all possible user interactions at the command line. At the end of the user interaction (the last simulated day), 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 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 not complete in all cases.

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 Abstract Factory, Singleton, and Command patterns are used.**

Capture all announcement output from a single simulation run and the 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 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.

Also include in your repository an updated version of the FNMS UML class diagram from part 1 that shows your actual class implementations in part 2. Note what changed between part 1 and part 2 (if anything) in a brief comment paragraph. Again – note where patterns are in use in the diagram.

**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, Damaged Items, 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 clear and 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 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 3/2 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 2 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 FNMS simulation program in Java with the new changes and patterns. This includes identifying major contributing or communicating classes (ex. Items, Staff, etc.) 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 50 points (plus possible 10 point bonus) and is due Wednesday 3/9 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, a captured Output.txt text file with program results, 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).

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 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 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 3/5 through the end of the day Thursday 3/10. 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.**

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.