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 Project 2 and 3. You may reuse code and documentation from your Project 2 or 3 submissions. You may also use example code from class solutions to Project 2. 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. 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/1IsR_bovtLqCrgW3EA0paFXHdyS_D-QmgMPoVUxMWBqo/edit?usp=sharing

2) (10 points) UML Sequence Diagram for Project 3. Select at least four top level active objects for your simulation (Ex. Cashier, 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 Cashier in a day. The sequence diagram can be 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 FLGS 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: FLGS simulation second extension – 50 points

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 a friendly local game store (FLGS). The simulation should perform all functions previously enabled in Project 3. Cashiers will continue to perform all functions they performed in Project 3. The simulation will be refactored with the following extensions.

1) There is a 1 in 30 chance that the store will be robbed after it closes on the first day. This event can only occur once. If the store is robbed all money, cookies, and games will be removed from the store. The store will be closed the next day while insurance pays for restocking all games, cookies, and money as if it was the first day. Damaged Games will not be stolen. Appropriate messages should be included for this event.

2) Create a common naming method that selects names from a circular list or by using a name creation algorithm. Use this naming method to name object instances of the new Customer and Demonstrator classes below.

3) Customers will be now created as a full class structure and unique object instances that will be used during the Open the Store Cashier event. Use a Factory Pattern to create each type of arriving Customer object. Customer types include Family Gamer, Kid Gamer, Card Gamer, Board Gamers, and Cookie Monster. Decide on the chance of each type of Customer arriving (the Cookie Monster should only rarely arrive on 1% or 2% chance). Customers, when created, should randomly assign base purchase chance bonuses to the 3 games in their category of 20%, 10%, and 0%. (Ex. a Board Gamer arrives with a 20% bonus chance to buy Risk, 10% for Gloomhaven, and 0% for Catan.)

4) The Announcer will become a Singleton Pattern object. Create two versions of the Announcer, one with eager instantiation (EagerAnnouncer), one with lazy instantiation (LazyAnnouncer). Clearly indicate how the version of the Announcer Singleton can be selected at compile or run time.

5) A new Employee, the Demonstrator, will be created. The Demonstrator will arrive at the store just before Customers begin entering and will leave after all Customers are served. Arriving Customer objects will make Command Requests to the active Cashier before purchasing Cookies or Games. Customers will use a Command Pattern to issue Command Requests to the active Cashier. The Cashier (as Invoker) will ask the Demonstrator to perform Demonstrate, Recommend, and Explain Commands. Remember that as an Employee, the Demonstrator must use the Announcer to say what it is doing.
   For Command Requests the Customer will:
   a. (25% of the time) ask someone to Demonstrate a random game of the type they are looking for
   b. (30% of the time) ask someone to Recommend a random game of the type they are looking for
   c. (20% of the time) ask someone to Explain a random game of the type they are looking for
   d. (25% of the time) enter the store without issuing a Command Request
   e. If the Demonstrator Demonstrates, Recommends, or Explains any game to a customer, the chance that customer will buy that instance of a game (Ex: Magic, Risk) increases by 10%; also if the Demonstrator performs a command, it should send that event string to the Announcer (Ex: “Dale the Demonstrator demonstrates Catan for Customer Carl”).
   f. The Customer will randomly issue 1 to 3 Command Requests on arrival (and will stop if they randomly decide to enter the store)
   g. If the Customer arriving is the Cookie Monster, the Demonstrator will scream and run away; a new Demonstrator with a new name should be instantiated for the next customer; the Cookie Monster will then enter the store as normal

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.

Simulate the running of the store for 30 days. As in Project 2 and 3, at the end of the 30 days, for each game type, list the number in inventory, the number sold, and the total sales for that game type. Also list the contents of the Damaged Game container, the final count of money in the Cash Register, as well as how many times money was added to the register due to low funds in the Count the Money step. Additionally, list the number of cookies sold each day and in total, as well as total cookies lost to the Cookie Monster; and the total amount of money paid to Gonger (which he puts in his pocket) for his cookies.

Capture all 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). Clearly identify test output as TestOutput.txt for review.

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

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 three lines – Game Sales, Cookie Sales, and Total Register $ – for each day of the simulation. Another graph should show three lines – Games in Inventory, Damaged Games, and Games 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, it 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 10/13 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 just based on your providing the 5 sections listed in your proposal (as well as making the Google Doc entry for the project).

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

Question 3 should provide a UML class diagram that could be followed to produce the FLGS simulation program for Project 4 in Java. This includes identifying major contributing or communicating classes (ex. Games, Employees) and any methods or attributes found in their design. As stated, multiplicity and accessibility tags are

optional, but patterns should be indicated. Use any method reviewed in class to create the diagram that provides a readable result, including diagrams from tools or hand drawn images. 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 10/20 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, 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 new patterns (Factory, Singleton, Command) to be illustrated in the code. A penalty of -2 to -4 will be applied for instances of poor or missing comments 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. A penalty of -1 to -3 will be applied per exercise for incomplete or missing output. This includes a separate output to verify test case results.

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 from part 1 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

Note that the class Midterm Exam runs from Sat 10/16 noon to Wed 10/20 midnight. Plan your schedules accordingly to complete both the exam and the team project. No extensions will be provided.

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.