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.

Part 1: OO and UML exercises – 20 points

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

1. (5 points) What does "design by contract" mean in an OO program? What is the difference between implicit and explicit contracts? Provide a text, pseudo code, or code example of each contract type.

2. (5 points) What are three ways modern Java interfaces differ from a standard OO interface design that describes only function signatures and return types to be implemented? Provide a Java code example of each.

3. (5 points) Describe the differences and relationship between abstraction and encapsulation. Provide a Java code example that illustrates the difference.

4. (5 points) Draw a UML class diagram for the FNMS simulation described in part 2. The class diagram should contain any classes, abstract classes, or interfaces you plan to implement to make the system work. Classes should include any key methods or attributes (not including constructors). Delegation or inheritance links should be clear. Multiplicity and accessibility tags are optional. Design of this UML diagram should be a team activity if possible.

Part 2: FNMS simulation – 30 points

This is the first of three music store projects. You'll reuse all or part of this code in projects 3 and 4.

Your team will create a Java program to simulate the daily operations of friendly neighborhood music stores (FNMS). A store contains a selected set of merchandise arranged in the following hierarchy of musically oriented merchandise items:

- Item (name, purchasePrice, listPrice, newOrUsed, dayArrived, condition, salePrice, daySold)
  - Music (band, album)
    - Paper Score
    - CD
    - Vinyl
  - Players
    - CD
    - Record Player
    - MP3
  - Instruments
    - Stringed (electric)
      - Guitar
      - Bass
      - Mandolin
    - Wind
      - Flute (type)
      - Harmonica (key)
  - Clothing

        - Hats (hatSize)
        - Shirts (shirtSize)
        - Bandanas
    - Accessories
        - Practice Amps (wattage)
        - Cables (length)
        - Strings (type)

You should use a multi-level inheritance structure for these objects as shown. If an object is followed by words in parenthesis, these are attributes particular to that subclass. You can determine values for any attributes not specified in the directions. For instance, each item needs a name, but you can determine how you want to name item instances.

The store keeps all money for buying merchandise and accepting money from Customers in a Cash Register. At the start of the simulation, the amount of money in the register is $0.

The store does have a starting inventory of merchandise items that should be initialized before the simulation starts. There are three of each lowest subclass of merchandise items (3 CDs, 3 Shirts, 3 Guitars, etc.) in the store at the start of the simulation. Determine a name (any way you choose) and a purchase price ($1 to $50) per item. The list price will be set to 2 x the purchase price for each item. The attribute dayArrived should be set to 0 for the initial item set. Condition has five levels: poor, fair, good, very good, or excellent. The attributes salePrice and daySold will be set when the item is sold.

The store has two Staff members of type Clerk, named Shaggy and Velma. (There will be other types and instances of Staff in the future.) One of the two of them will randomly report for work each day the store is open during the simulation, but neither will work more than three days in a row. The store does not open on Sundays, so 1 day of each 7 the store will not open (this should be announced).

On each day the simulation runs when the store is open, the following Actions will be taken by the Clerk that is working that day:

ArriveAtStore: The Clerk will announce their arrival at the FNMS. Announcements should be text output to the console and should look something like: “Velma arrives at the store on Day 4.” The Clerk may find items waiting that have been delivered from a prior day (see the PlaceAnOrder action). These items should be put into the inventory of merchandise items for the store and these added items should be announced.

CheckRegister: The Clerk will count and announce the amount of money found in the register. If there is insufficient money in the register, the Clerk must perform the GoToBank Action next before going on to the DoInventory Action.

GoToBank: This Action is only performed if there is less than $75 in the register. The Clerk will go to the bank, withdraw $1000, and put the money in the store register with an announcement. The amount of money withdrawn from the bank in this manner should be tracked.

DoInventory: The Clerk will add up the value of all the items in the store (based on purchase price) and announce that total value. If any of the item subclasses has a count of 0 (for instance, there are 0 CD player items in inventory), the Clerk must perform the PlaceAnOrder Action for each missing item type before going on to the OpenTheStore Action. All activity should be announced.

PlaceAnOrder: For any subclass item that is at 0 inventory, the Clerk will order 3 of those items, each with a random purchase price and other randomly determined characteristics (similar to adding items on Day 0). The purchase price of each of these items should be paid for by removing the funds from the Cash Register. These items should arrive at the store in the next 1 to 3 days, to be found by the Clerk in the ArriveAtStore step, at which point the Clerk will put them into the store inventory. All activity should be announced.

OpenTheStore: The Clerk will now respond to arriving customers. Customers will come in to either buy or sell a single item. There will be 4 to 10 buying Customers and 1 to 4 selling Customers each day. All actions taken by Customers should be announced.

> A Customer that wants to buy an item will be looking for a random item type from the subclasses of items (ex. Banjo or Cables). If there are no items of the type they want to buy, they will leave without buying. If there is an item of the type they want to buy, there is a 50% chance they will pay the listPrice for the item. If they do not buy the item, the Clerk can offer them a 10% discount to the listPrice. There will then be a 75% chance they will buy the item at that price. When an item is sold it should be moved from the store inventory into a Sold Items collection, and the daySold and salePrice should be updated. The money paid by the customer should go into the Cash Register.
>
> A Customer that wants to sell an item will come in the store with a random item type. The Clerk will look the item over and determine the item attributes newOrUsed, condition, and a purchase price. Purchase price will be based on condition: each of the five levels of condition should have its own random price range that increases by condition. The Clerk will offer the Customer the purchasePrice they determine. The customer has a 50% chance of accepting that price and selling the item. If they choose not to sell the item, the Clerk will add 10% to the offered price. At that point, the customer will have a 75% chance of selling the item to the store. When an item is sold to the store, it should be added to the store inventory (setting all values as appropriate), and the purchasePrice should be paid from the Cash Register to the customer.
>
> Announcements should make clear what happened:
>
> “Shaggy sold a CD to Customer 3 for $13 after a 10% discount.”
>
> “Velma bought a poor condition used guitar from Customer 9 for $33.”
>
> “Customer 6 wanted to buy a flute but none were in inventory, so they left.”

CleanTheStore: Before closing, the Clerk will clean the store. It’s possible an item could be damaged during cleaning. Velma is a careful cleaner, and only damages a random item in inventory 5% of the time. Shaggy is less careful and damages a random item 20% of the time. When an item is damaged, its condition is lowered one level, and the listPrice is reduced 20%. If the item was already in Poor condition, it is removed from inventory. Anything that occurs in this phase should be announced.

LeaveTheStore: The Clerk will lock up the store and go home for the day (which should be announced).

Simulate the running of the FNMS store for 30 days. At the end of the 30 days, print out a summary of the state of the simulated store, 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, point out at least one example of: Inheritance, Polymorphism, Cohesion, Identity, Encapsulation, and Abstraction.

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).

Also include in your repository an updated version of the FNMS UML 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 comment paragraph.

Grading Rubric:

Homework/Project 2 is worth 50 points total.

Part 1 is worth 20 points and is due on Wednesday 2/2 at 8 PM. The submission will be a single PDF per team. The PDF must contain the names of all team members.

Questions 1-3 will be graded on the quality (not quantity) of the answer. Questions one and two should include examples and citations. Solid answers will get full points, poor-quality or missing elements for answers will cost -1 or -2 points each, missing answers completely will be -5 points.

Question 4 should provide a UML class diagram that could be followed to produce the FNMS simulation program in Java. 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. 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 30 points and is due Wednesday 2/9 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, and a README file that has the names of the team members, the Java version, and any other comments on the work – including clearly documenting any assumptions or interpretations of the problem. Only one URL submission is required per team.

10 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 six OO terms to be illustrated in the code. A penalty of -1 to -2 will be applied for instances of poor or missing comments or excessive procedural style code (for instance, executing significant procedural logic in main).

5 points for correctly structured output: The output from a run captured in the text file mentioned per exercise should be present. A penalty of -1 to -2 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 about your implementation or assumptions 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, including how the structure changed between parts 1 and 2. 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.