Problem Domain: Specialty Food Business Simulator

For this homework, you will design an object-oriented program to model the following problem domain.

You are responsible for the point of sale/inventory system for a gourmet food store chain. At present there is only one location, but eventually there will be several others.

The store has a catalog of five different specialty foods it sells: spring rolls, egg rolls, pastry rolls, sausage rolls, and jelly rolls. The store begins the first day with the inventory to make 30 of each type of roll. Whenever a store makes the last of a given type of roll, that roll may no longer be sold that day. If all roll inventory runs out, the store will close for that day, and a notice of closure for no inventory should be raised. When the store sells the last of a given type of roll, an inventory out notice will be raised, and the next day, 30 more rolls of that type will be added to the production capacity. Note that inventory is only set back to 30 the next day if the inventory ran completely out the prior day. You should decide the cost of each roll (each should be different).

Whenever a customer buys a roll, they may optionally ask for extra sauce, extra filling, or extra topping. Each of these options has a different cost for each different roll type (you can decide the cost of each). You do not need to maintain a capacity or inventory of these extras. The store always has enough.

Customers will arrive randomly every day in random order. There are three types of customers: casual, business, and catering. Casual customers will buy 1 to 3 rolls (type determined randomly). Business customers will buy 10 rolls, two of each type. Catering customers will buy 5 rolls of 3 different types (15 total, type determined randomly). A total of 1 to 12 casual customers, 1 to 3 business customers, and 1 to 3 catering customers will arrive each day. If the store closes at any point during the day due to no roll inventory, any customers that would have arrived after that point will not go to the store.

For every individual roll ordered by any customer type, there is a chance of adding 0 to 3 extra sauces, 0 to 1 extra fillings, and 0 to 2 extra toppings. The cost of these extras should be added to the roll order total cost.

Responses to roll outages by customers will be different. A casual customer will try to select a different roll type to make up their 1 to 3 roll order if the initial one they select is out for the day. A casual customer will take any number of available rolls (if they wanted 3, but only 2 were available, they will take them). A business customer will only take their order if it is filled exactly as requested, otherwise, they will not make a purchase. A catering customer will take any number of available rolls of any available types up to a total of 15, if their original order cannot be placed. If the store is open, and the original order is not filled in any way, this should be counted for the day by customer type (e.g. 2 casual customers and 1 business customer did not fill their original orders on day 4 due to roll outages).

The store will maintain sales data, inventory levels, costs of orders, and special events for daily display and summary statistics.

Assignment

Write an object-oriented program in Java 8 or later that implements the problem domain and does the following:

Simulates 30 days of store and customer activity.

The purpose of this assignment is NOT to meet the requirements by any means necessary. A program that does the simulation above and produces the requested output but makes use of only structured or functional programming techniques (i.e. no objects, just data structures and a main program) will receive zero points (for the whole assignment). Only object-oriented programs that show good use of abstraction, encapsulation and polymorphism and meet the above requirements will be able to get full credit for the program.

Further, you should select and use at least three distinct OO patterns that we have discussed in your design. Typical uses might include factory (for instantiating rolls or customers), decorator (for adding extras), observer (for capturing events that will go to output). You may see other opportunities for other OO pattern use. Document and comment clearly where any patterns are applied.

Also include at least 10 JUnit test methods (methods with the @Test decorator) in a single test class called MyUnitTest. These unit tests should be applied appropriately to classes and methods in your simulation code to ensure their proper behavior. You should be able to instantiate a MyUnitTest object before your simulation run and then execute each of the ten test methods at some point before, during, or after the simulation, with output from each test indicating the identification of the test and success or failure of each test method. A single test method may contain single or multiple assert statements as you feel are needed or are appropriate. Clear output from the test run must be included in your program results.

I suggest using a simple test class containing test methods using a variety of assert statements as outlined in:

- http://tutorials.jenkov.com/java-unit-testing/simple-test.html
- http://tutorials.jenkov.com/java-unit-testing/asserts.html
- http://tutorials.jenkov.com/java-unit-testing/matchers.html

I further suggest using a later version of JUnit 4, but that is up to you. If you or your team are more experienced with JUnit, you may want to implement the tests in a more sophisticated fashion. As long as at least ten test methods are created and clearly applied, you may use any alternate implementation approach.

For each day the simulation runs, the program should print the following information in a concise format:

- Day number
- Rolls in inventory at the start and finish of each day, by type
- Roll sales – individual customer orders: including customer type, number of rolls by type, any added extras per roll, and the total cost of the order
- Total payment for orders for the day by customer type, and overall
- Count of orders impacted by a roll outage, by customer type (do not count customers that did not go to the store after it closed)
- Count inventory orders by roll type
- Indicate if the store closed for no inventory that day

At the end of 30 days, print (concisely):

- Total number of rolls sold, by type and overall
- Total money in sales
- Total number of roll outage impacts

Re-run the program two more times increasing the inventory level at the start and at replacement from 30 rolls to 45 and then from 45 to 60.

Grading Rubric

As always, your OO code should be well structured and commented, and citations and/or URLs should be present for any code taken from external sources. Comments should focus on what is being done in the code, not on how, unless the method in question is unclear or obscure. You may not directly use code developed by other student teams, although you may discuss approaches to the problems, and may wish to credit other teams (or class staff) for ideas or direction.

The submission is a GitHub Repository URL. The Repo must contain:

- Running, commented OO code for the simulation in Java 8 or later
- A README Markdown file with
  - Program title and the names of team members
  - Any special instructions to run the code (graders may request demonstrations)
  - Language and environment used for development
  - Text description of program design, including any assumptions you have made about specifications that may not be clear or complete
- A text output file containing a capture of complete output from the full simulation run including test cases
- A PDF containing a full UML Class diagram that shows classes and relationships from your design (include data attributes and key methods in the class diagram, as well as identification of any pattern support in the class structure)

Homework/Project 3 is worth **75 points** as follows. There is also an additional **10 point** extra credit opportunity (see below).

- 10 Points – README file (-1 or -2 for incomplete or missing elements)
- 10 Points – PDF with UML class diagram (-1 to -2 for poor quality elements or lack of pattern identification, -10 for missing UML diagram)
- 15 Points – Use of at least 3 patterns in simulation design (-1 or -2 for implementation issues or poor commenting per pattern, -5 if a pattern is missing)
- 20 Points – Execution as evidenced by text-based output file (-2 or -4 for missing expected functionality or output elements)
- 20 Points – Code quality (-1, -2, -3 for issues including poor commenting, procedural style code, or logic errors; -20 if not an object-oriented solution as requested).

Any UML tool can be used to create the UML class diagram. If done on paper/pencil or whiteboard, please be sure the diagrams are readable and clear for grading.

Homework/Project 3 is due at noon on Wednesday 10/14.

Assignments will be accepted late as follows. There is no late penalty within 4 hours of the due date/time. In the next 44 hours, the penalty for a late submission is 5%. In the next 48 hours after that the late penalty increases to 15% of the grade. After that point, assignments will not be accepted.

Late penalties will be waived for health or medical related issues (of course).

Use office hours, e-mail, or Piazza to reach the class staff regarding homework/project questions.

For 10 points extra credit, present 4 Java-based graph/chart images of the simulation data:

- For each 30 day run at 30, 45, 60 roll inventory levels, show a line graph of daily sales and daily roll outage (3 line graphs)
- Create one comparison bar graph of the 3 summary measures for the 30, 45, and 60 roll inventory levels: total number of rolls sold overall, total money in sales, total roll outage impacts
- These should be generated with a Java chart library of your choice, possibly JFreeChart, XChart, or a similar library
- The images can be captured for storage in the repo in any way you like if they are easily reviewable (PDFs are appreciated)