CCPROG3 MCO Specifications

Program Overview – Vending Machine Factory Simulator

Vending or "automatic retailing" has its roots traced back as far as 215BC. However, it was only in the early 1880s that the first commercial coin-operated vending machines were introduced in London, England which dispensed postcards. Now, modern vending machines offer more variety of products ranging from drinks, tickets, snacks, WIFI connections, and many more. In Japan, vending machines are designed to function during blackouts, particularly in the wake of earthquakes and aftershocks, thus utilizing vending machines as an emergency aid.

In an effort to invoke an appreciation of the many fascinating automation wonders of our modern era, you will re-create a vending machine. The final product for your project is to create a program that simulates a Vending Machine Factory. In your menu-based program, there must be options to do the following until the user chooses to exit the program:

1. **Create a Vending Machine**
   In this feature, the user is asked to choose to create either a Regular or a Special Vending Machine. Descriptions of the basic attributes and behaviors of both vending machines are described below. For this option, the program does not need to maintain any previously created Vending Machines.

2. **Test a Vending Machine**
   In this feature, the user is asked to choose to either test the Vending Features or the Maintenance Features of the current vending machine (i.e., the most recent that was created). When Vending Features are chosen, the different options are tested until the user chooses to end Vending Features Test, which will then bring the user back to the Test a Vending Machine menu options. The same is done when the user chooses Maintenance Features.

3. **Exit**
   In this feature, the program terminates properly.

A **Regular Vending Machine** consists of item slots that act as an interface for the user to know what is available for purchase. Each slot is mapped to a specific item and – to keep things simple – kindly assume that items stored in the slots are unique from each other. The vending machine does not hold infinite items, so there is a common limit for the number of items that can be stocked. For the project, there must be at least eight (8) slots and the vending machine should have a capacity of at least ten (10) items per slot[1]. The availability of an item should be obvious to the user. In terms of vending features, the machine should be allowed to receive payment from the user in different denominations[2], dispense the item based on the choice of the user, and produce change. Note that a user may proceed directly with producing change and skip choosing an item – as if they changed their mind about making a purchase. Additionally, if there is not enough change in the machine, a transaction should not take place and the user should be informed of the issue. Note that all vending machines must also inform the user of the amount of calories found in the item. Lastly, as vending machines do not hold infinite items, you may assume that the owner/operator of the vending machine regularly performs maintenance. Maintenance features include restocking/stocking specific items, setting the price of each item type, collecting the payment/money, and replenishing the money (for different denominations) that will be used by the machine to provide change. Also, the vending machine has the capability of printing a summary of transactions. In other words, the vending machine should at least list the quantity of each item sold and the total amount collected in the sales starting from the previous stocking. This implies that there should also be a display of the starting inventory and the ending inventory from the last restocking.

Special Vending Machines can also be produced by the simulation program. These machines are special because, apart from the features of a regular vending machine, the machine can also prepare a selected product based on (a) the items that are stored in the machine and (b) the choices of items for the product that the user wants. This means that the amount of calories for the final product is the combination of the calorie count of each chosen item to include (which might involve more than just addition). Note that since this is a simulation of the machine's work, your program will display how the final product is "prepared".

[1] You may choose to make the capacity of the vending machine larger, but the vending machine must not have smaller numbers than indicated.
[2] As if the user were inserting coins or bills of different denominations into the machine.

As an example, imagine a vending machine that can dispense a customizable ramen, on top of other items. The owner of this particular ramen vending machine (RVM) will also set the items that can be chosen in the customizable ramen and the prices for each. In this case, items that can be included in the customizable ramen are noodles, egg, chashu pork, fried tofu, negi, tonkotsu broth, ukokkei broth, miso broth, and shio broth. Please note that another owner may choose to stock the following (on top of what is previously indicated): different types of noodles, boiled chicken slices, fish cake, and black garlic oil.

Keeping the RVM example in mind, we can imagine that a customer can order the following customized ramen: 1 order of noodles, 2 orders of chasyu pork, and 1 order of tonkotsu broth. During the preparation, the machine will display something like this:

```
Blanching noodles…
Heating broth…
Placing noodles in cup…
Topping with chashu pork…
Pouring broth…
Ramen Done!
```

Another customer may order a customizable ramen having 2 orders of noodles, 1 order of aji tamago, fried tofu, and miso broth and the display will be something like this:

```
Blanching noodles…
Heating broth…
Putting noodles in cup…
Topping with fried tofu and aji tamago…
Pouring broth…
Ramen Done!
```

It should be noted that another customer may only order chashu pork from the RVM. In this case, this is treated as an item. Therefore no “preparation” is necessary and the item is dispensed directly. However, some items that do not make sense to stand alone as a food item should not be allowed to be sold separately. For example, black garlic oil may not be chosen as an item and is only available when choosing add-ons to the customizable ramen.

As is the practice, one transaction in either of the vending machines will dispense one (1) item only.

The teacher may choose to dictate the customizable products or allow the student to decide for themselves. Some other suggested customizable products:

1. Sisig[3]
2. Silog meals
3. Pinoy burger
4. Halo-halo
5. Ice scramble

Please take note, however, that the special vending machine should have products that need to be assembled using other items, items that can be purchased individually, and items that are not meant to be sold to the user.

MCO1 Requirements

1. The program is a simulator of Vending Machine Factory for Regular Vending Machines ONLY.
   - Consequently, there are also no Customizable Items/Products in Phase 1.
2. In the implementation and during the demo[4], the proponents are to identify and represent items (and corresponding attributes) that are to be placed in this vending machine that are related to or are relevant to the phase 2 customizable product.
   - For example, if phase 2 product is ramen, then phase 1 products could be fish cake, aji tamago, fried tofu, among others.
   - Additionally, a breakdown of planned items for MCO2 is expected as a separate submission. These include items that are to be created by the special vending machine using other items (+ a breakdown of the stock per item needed), items that can be sold individually, and items that cannot be sold individually. Not all items are expected to be part of the design and implementation of MCO1 since only the regular vending machine is expected; however, this requirement will help align the design of your MCO1 with the MCO2.

[3] Do not worry about being judged by the Kapampangans.
[4] For the MCO1 demo, your instructor will specify the mode of delivery.

3. Design and implementation should exhibit proper object-based concepts, like encapsulation and information hiding.
4. Text-based output simulation of the features, including showing internal processing.
   - For example, indicating “Dispensing Item”, “Dispensing change with the following denominations…”) and error messages, such as “Not enough money to produce change, please insert the exact amount to continue the transaction.”).
   - No GUI is expected for MCO1

MCO2 Requirements

1. The program is a simulator of Vending Machine Factory of both types of machines.
2. Implementation should simulate real-world objects.
   - That is, a stock of 10 aji tamago is not just an attribute quantity for the item, but there are actually 10 instances of the item and each is “destroyed” when dispensed or used as an ingredient in the customizable product.
3. Design and Implementation should exhibit proper object-oriented concepts, like inheritance, polymorphism, and method overloading/over-riding.
4. GUI-based output simulation of the features of the factory, as well as the vending machine.
5. The program should also utilize the MVC architecture.

General Instructions

Deliverables
The deliverables for both MCOs include:
1. Signed declaration of original work (declaration of sources and citations may also be placed here)
   - See Appendix A for an example
2. Softcopy of the class diagram following UML notations
   - The class diagram should exhibit the appropriate object-oriented programming principles
   - For MCO1, the class diagram only needs to depict the requirements for MCO1
   - For MCO2, the class diagram should be complete and only depict the Model components
3. Javadoc-generated documentation for proponent-defined classes with pertinent information
4. Zip file containing the source code with proper internal documentation
   - The program must be written in Java
   - Test script following the format indicated in Appendix A

Submission
All deliverables for the MCO are to be submitted via **Canvas**. Submissions made in other venues will not be accepted. Please also make sure to take note of the deadlines specified on Canvas. No late submissions will be accepted.

Grading
For grading of the MCO, please refer to the MCO rubrics indicated in the syllabus or in Appendix B and C of this document.

Collaboration and Academic Honesty
This project is meant to be worked on as a pair (i.e. max of 2 members in a group). In exceptional cases, a student may be allowed by their instructor to work on the project alone; however, permission should be sought as collaboration is a key component of the learning experience. Under no circumstance will a group be allowed to work on the MCO with more than 2 members.

A student cannot discuss or ask about design or implementation with other persons, with the exception of the teacher and their groupmate. Copying other people’s work and/or working in collaboration with other teams are not allowed and are punishable by a grade of 0.0 for the entire CCPROG3 course and a case may be filed with the Discipline Office. In short, do not risk it; the consequences are not worth the reward[5].

Documentation and Coding Standards
Do not forget to include internal documentation (comments) in your code. At the very least, there should be an introductory comment and a comment before every class and every method. This will be used later to generate the required External Documentation for your Machine Project. You may use an IDE or the command prompt command javadoc to create this documentation, but it must be PROPERLY constructed.

[5] What is a measly passing grade compared to a life-long burden on your conscience?

Please note that we're not expecting you to add comments for each and every line of code. A well-documented program also implies that coding standards are adhered to in such a way that they aid in the documentation of the code. Comments should be considered for more complex logic.

Bonus Points
An MCO2 submission can receive at most 10 bonus points based on additional features if and only if the submission has met the basic requirements (i.e. exemplary for program correctness and design of classes and their relationships). Bonus points will only apply to MCO2.

Awarded points may vary based on the complexity of the additional feature. Assets that make the MP more engaging or that add to the overall user experience may also be awarded points. Groups have the freedom to add bonus features as they see fit. However, added features must not permanently alter the base requirements of the project. There should always be a quick and easy way to test the minimum requirements in the final submission, despite any added features. If a bonus feature needs to alter the base requirements, the group must provide a setting that will momentarily disable such alterations. Not being able to test the expected base requirements will most likely result in deductions to the submission's score, as well as the bonus points not being counted.

To encourage the usage of version control (such as Git), up to 4 bonus points may be awarded. While the usage of version control will not be taught in this course, you are encouraged to organize and store your code via a Git repository, like services offered by GitHub. Utilizing some form of version control will make it easier for the members of the group to collaborate with each other and the commit history also helps in providing some form of accountability. Awarded points may vary based on how the group was able to leverage the usage of version control (e.g. small and often commits, descriptive commit comments, contributions from all members, branching).

Note that bonus points are capped at 10 points. For example, if a group was awarded 10 points for the implementation of additional features and 4 points for effective usage of version control, only 10 points will be awarded. Lastly, do not waste your effort on additional points when the project requirements have not been met!

Resources and Citations
We would just like to emphasize that you DO NOT need to create your own sprites or images (background pictures, item/product pictures, etc.) for the simulation. You can just use what's available on the Internet and just include them in your project. If you wish to do so, we would also like to remind your to please cite where you got your sprites.

All sources should have proper citations. Citations should be written using the APA format. Examples of APA-formatted citations can be seen in the References section of the syllabus. You're encouraged to use the declaration of original work document as the document to place the citations.

Demonstration
The mode of delivery of the demo for MCO1 is left to the discretion of your instructor. However, MCO2's demo is expected to be conducted live (whether F2F or live). During the demo, all members of the group should be present and follow the instructions given by the instructor. Apart from Q&A, a demo problem will be given to the group as part of the demo.

A student or a group who is not present during the demo or who cannot answer questions regarding the design and implementation of the submitted project convincingly will incur a grade of 0 for that project phase.

During the MP demo, it is expected that the program can be compiled successfully and will run. If the program does not run, the grade for that phase is 0. However, a running program with complete features may not necessarily get full credit, as implementation (i.e., code) will still be checked.

Other Notes
You are also required to create and use methods and classes whenever possible. Make sure to use Object-Based and Object-Oriented Programming concepts properly. No brute force solution.

Statements and methods not taught in class can be used in the implementation. However, these are left for the student to learn on his or her own.

Appendix A. Template for Declaration of Original Work

Declaration of Original Work

We/I, [Your Name(s)] of section [section], declare that the code, resources, and documents that we submitted for the [1st/2nd] phase of the major course output (MCO) for CCPROG3 are our own work and effort. We take full responsibility for the submission and understand the repercussions of committing academic dishonesty, as stated in the DLSU Student Handbook. We affirm that we have not used any unauthorized assistance or unfair means in completing this project.

[*In case your project uses resources, like images, that were not created by your group.*] We acknowledge the following external sources or references used in the development of this project:

1. Author. Year. Title. Publisher. Link.
2. Author. Year. Title. Publisher. Link.
3. Author. Year. Title. Publisher. Link.

By signing this declaration, we affirm the authenticity and originality of our work.

Signature and date	*Signature and date*
Student 1 Name	Student 2 Name
ID number	ID number

[*Note to students: Do not submit documents where your signatures are easily accessible. Ideally, submit a flattened PDF to add a layer of security for your digital signatures*]

Appendix B. Example of Test Script Format

Method	**#**	**Test Description**	**Sample Input Data**	**Expected Output**	**Actual Output**	**P/F**
isPositive	1	Determines that a positive whole number is positive	74	true	true	P
	2	Determines that a positive floating point number is positive	6.112	true	true	P
	3	Determines that a negative whole number is not positive	-871	false	false	P
	4	Determines that a negative floating point number is not positive	-0.0067	false	false	P
	5	Determines that 0 is not positive	0	false	false	P

Appendix C. Rubrics for MCO1

Total: 100 points

Criteria	Exemplary	Satisfactory	Developing	Beginning	None
[Prerequisite] **100%**					Late submission of deliverables. OR Part or all of deliverable is plagiarized or not a product of student's output. OR No significant contribution to the group output. OR Did not appear during the demo. **0**
Program Correctness and Completeness	Program executes without errors, and properly provides all the functionalities of the object-based implementation. **40**	Program executes without errors, but is missing some minor features. **30-35**	Program executes with minor errors, or is missing significant features. **20 - 25**	Program executes with minor errors and is missing significant features. **10 - 15**	Program does not run OR Program does not produce any output. **0**
Designing Classes/Objects	Design decisions comply completely with the specs and all classes, attributes, behaviors, and relationships identified and shown in the UML class diagram are logical.	Design decisions comply completely with the specs as shown in the UML class diagram, but include unnecessary or redundant classes OR there is incomplete information on the attributes, behaviors, or relationships.	The design provides for most but not all of the original specs as shown in the UML class diagram. Most likely, include unnecessary or redundant classes AND there is incomplete information on the attributes, behaviors, or relationships.	The UML class diagram does not include most information based on the specs.	No submitted UML class diagram. OR Design of classes are not in compliance to a logical object-based design.

	20	15	10	5	0
Consistency of design and code	Program implementation corresponds correctly with the presented Object-based design.	There are minor inconsistencies in design or implementation of attributes or methods, but all necessary features are still provided.	There are minor inconsistencies in design or implementation of attributes or methods, which leads to missing or unexpected features.	There are significant inconsistencies between design and implementation at the class level, but most features are still apparent.	No submitted class diagram to compare with.
	10	**8**	**5**	**3**	**0**
Program Readability and Documentation	Coding standards prescribed in the course is followed. AND In-line comments are included for long codes, apart from proper documentation of methods (inclusive of pre-conditions, post-conditions, method parameters, and return data) via Javadoc.	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. But, there is proper documentation of methods (inclusive of pre-conditions, post-conditions, method parameters, and return data) via Javadoc.	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. There are method documentation via Javadoc, but are missing some information.	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. AND Method documentation is not via Javadoc annotation.	No internal documentation.
	10	**8**	**5**	**3**	**0**
Test Case Design and Documentation	Apart from getters and setters, all methods in all classes are tested with at least 3 documented unique test cases.	All methods in all classes are tested, but not all have at least 3 documented unique test cases.	Most methods in all classes are tested with at least 3 documented unique test cases.	Many methods do not have at least 3 documented unique test cases.	No test script submitted.
	10	**8**	**5**	**3**	**0**
Delivery and Presentation	All members are present and on-time. All deliverables are in the correct format. Members exhibit mastery of their project through the demo and Q&A.	All members are present and on-time. All deliverables are in the correct format. Members exhibit familiarity of their project through the demo and Q&A.	All members are present. Some deliverables are in the incorrect format. Members exhibit familiarity of their project through the demo and Q&A.	All members are present. Some deliverables are in the incorrect format. Members exhibit some knowledge of their project through the demo and Q&A.	Did not appear during the demo. OR Member did not exhibit ample knowledge about the design AND implementation of

	10	**8**	**5**	**3**	the project. **0**

Appendix D. Rubric for MCO2

Total: 100 points

Criteria	Exemplary	Satisfactory	Developing	Beginning	None
[Prerequisite] **100%**					Late submission of deliverables. OR Part or all of deliverable is plagiarized or not a product of student's output. OR No significant contribution to the group output. OR Did not appear during the demo. **0**
Program Correctness and Completeness	Program executes without errors, and properly provides all the functionalities of the object-oriented implementation. **40**	Program executes without errors, but is missing some minor features. **30-35**	Program executes with minor errors, or is missing significant features. **20 - 25**	Program executes with minor errors and is missing significant features. **10 - 15**	Program does not run OR Program does not produce any output. **0**
Designing Classes/Objects	Design decisions comply completely with the specs and all classes, attributes, behaviors, and relationships identified and shown in the UML class diagram are logical. AND	Design decisions comply completely with the specs as shown in the UML class diagram, but include unnecessary or redundant classes OR there is incomplete information on the attributes, behaviors, or relationships. AND	The design provides for most but not all of the original specs as shown in the UML class diagram. Most likely, include unnecessary or redundant classes AND there is incomplete information on the attributes, behaviors, or relationships. OR	The design provides some of the original specs and includes unnecessary or redundant classes. AND Design is not following Model-View Controller.	No submitted UML class diagram. OR Design of classes are not in compliance to a logical object-oriented design.

	Design is following Model-View Controller. **15**	Design is following Model-View Controller. **10**	Design is not following Model-View Controller. **6**	**3**	**0**
Designing Class Relationships	Design decisions comply completely with the specs and all class relationships are logical and provide scalability to the system, as depicted in all information in the UML class diagram. **5**	Design decisions comply completely with the specs and all class relationships are logical, as depicted in complete information in the UML class diagram. **4**	The design provides for most but not all of the original specs, or some some information is missing in the UML class diagram. **3**	The design provides for most but not all of the original specs or some class relationships are unnecessary or redundant. **2**	No submitted UML class diagram. OR Design of relationships are not in compliance to a logical object-oriented design. **0**
Consistency of design and code	Program implementation corresponds correctly with the presented OO design. **5**	There are minor inconsistencies in design or implementation of attributes or methods, but all necessary features are still provided. **4**	There are minor inconsistencies in design or implementation of attributes or methods, which leads to missing or unexpected features. **3**	There are significant inconsistencies between design and implementation at the class level, but most features are still apparent. **2**	No submitted class diagram to compare with. **0**
GUI Usability	The Graphical User Interface is user-friendly and shows mastery and proper use of the API. **5**	The Graphical User Interface is usable and shows mastery and proper use of the API. **4**	The Graphical User Interface is usable but features some improper use of a few API components. **3**	The Graphical User Interface works properly but is difficult to use. Also, it exhibits some improper use of a few API components. **2**	Some of the GUI does not work properly. **0**
Program Readability and Documentation	Coding standards prescribed in the course is followed. AND In-line comments are included for long codes, apart from proper documentation of methods (inclusive of pre-conditions, post-conditions,	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. But, there is proper documentation of methods (inclusive of pre-conditions, post-conditions,	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. There are method documentation via Javadoc, but are missing some information.	Coding standards prescribed in the course is followed. AND No in-line comments are included for long codes. AND	No internal documentation.

	method parameters, and return data) via Javadoc. **10**	method parameters, and return data) via Javadoc. **8**	 **5**	Method documentation is not via Javadoc annotation. **3**	 **0**
Test Case Design and Documentation	Apart from getters and setters, all methods in all classes are tested with at least 3 documented unique test cases. **10**	All methods in all classes are tested, but not all have at least 3 documented unique test cases. **8**	Most methods in all classes are tested with at least 3 documented unique test cases. **5**	Many methods do not have at least 3 documented unique test cases. **3**	No test script submitted. **0**
Program Debugging	Solved the demo problem successfully in the given duration. **10**	Minor errors or discrepancies in expected output resulted after testing the solution to the demo problem. **8**	Some cases were not considered in the solution to the demo problem, thus resulted to errors in the result. **5**	Many cases were not considered in the solution to the demo problem. **3**	Did not appear during the demo. OR Cannot solve the demo problem in the given duration. **0**