CCPROG3 MCO1 Program Specifications

Program Overview – Trading Card Inventory System

Note: *MCO1 and MCO2 is to be done by pair, and by the same pair of students barring any reports of freeloading or decision by the pair to split. In the event of a split for any reason, the students who used to be in the pair would now need to work solo; they may not join an existing group or pair up with a new partner.*

Your task for MCO1 is to create a Trading Card Inventory System (TCIS). Card collectors often have difficulty tracking, organizing and trading cards from their ever-growing collections. Prices of cards also change all the time, so making sure trades are fair in terms of card value can be overwhelming or even intimidating for some. Your task is to create a simple TCIS with the following functionalities:

1. **Add Card**
   This feature allows a collector to add a card to their collection. In this feature, the user can input the details of a card they want to add to their collection. These details include:
   a. Card name – the name of the card. Card names are typically composed of letters but may also contain special characters. Card names are also unique, meaning if a card is added to the collection, and its name matches another card that is already in the collection, the user should be notified, and the user should be given the option to increase the count of the card instead (see **Increase/Decrease Card Count**).
   b. Rarity – cards can be common, uncommon, rare or legendary. Rarity should not be confused with Variant; only rare and legendary cards have variants.
   c. Variant – rare and legendary cards can be normal, extended-art, full-art or alt-art. If a card is not rare or legendary, their variant is always normal. Card variants influence the price of the card:
      i. Normal – no increase in value.
      ii. Extended-art – value is increased by 50%.
      iii. Full-art – value is increased by 100%.
      iv. Alt-art – value is increased by 200%.
   d. Value – corresponds to the dollar value of a card. In the case of cards with variants, the user is to input only the base value, not the increased value because of the card's variant.

2. **Increase/Decrease Card Count**
   Collectors usually keep more than one copy of a card. When a card is added to the collection, its count is automatically set to 1, but the collector may increase or decrease this count. If the count of a card becomes 0, the card should not be removed from the inventory, but the card should not be selectable for **adding to decks** or **adding to binders**.

3. **Display a Card/Display Collection**
   Collectors should be able to view the details of a card. By selecting a card via its name (or perhaps by the card's number in the collection), the rest of its details should be displayed on screen. The collector may also view their whole collection. Only the names of the cards in their collection and the count of each card must be displayed, sorted alphabetically in ascending order by card name.

4. **Manage Binders**
   Collectors usually put their cards in binders for trading. Your TCIS should allow collectors to:
   a. **Create a new Binder** – a new binder has a name and can hold up to twenty cards; that is, a binder has twenty slots. Note that multiple cards of the same name can be put in the same binder; each card will occupy its own slot.
   b. **Delete a Binder** – a collector must be able to delete a binder from the TCIS. If they do, all cards currently in the binder will be transferred back to the collection of the collector.
   c. **Add/Remove Card to/from the Binder** – cards can be added or removed from a binder. If a card is added to a binder, it is removed from the collection of the player. Should a card be removed from a binder, it is returned to the collection of the player.
   d. **Trade Card** – cards can be traded with other cards. For the purposes of MCO1, trades can only be one card for one card. In a trade, the collector selects a card from their binder to give up (called the **outgoing card**) and will receive a card (called the **incoming card**) from outside their binder (and indeed outside their collection) in return. Since the card is not from the collection, it should be added to the collection first (see **Add Card**). Once added,

the values of the incoming and outgoing card are compared. The collector will then be notified if the difference is greater than or equal to a dollar. The collector at this point should be given the option to cancel the trade. If they do, the incoming card is not added to the collection, and the outgoing card returns to the binder. If they do not cancel, or the trade proceeds normally because the values of the cards to be traded have a difference of less than a dollar, then the incoming card takes the slot of the outgoing card in the binder.

e. **View Binder** – display as a list all the cards in a selected binder. Similar to displaying the whole collection, the list of card names in the binder must be displayed, in alphabetical order. If two or more cards with the same name are in the binder, then the name of the card would be listed multiple times, depending on how many cards of that name are in the binder.

5. **Manage Decks**
   Trading cards are often also game pieces. While **you are not going to program the game itself**, you should allow collectors to manage and keep track of their decks. Your TCIS should therefore allow collectors to:
   a. **Create a new Deck** – a deck has a name and can hold up to ten cards. No more than one copy of a card can be in a deck (i.e. a card bearing the same name as another card in the deck may not be added to said deck but may be added to a different deck).
   b. **Delete a Deck** – like deleting binders, if a deck is deleted, the cards in the deck are returned to the collection.
   c. **Add/Remove Card to/from the Deck** – cards can be added or removed from a deck. If a card is added to a deck, it is removed from the collection of the player. Should a card be removed from a deck, it is returned to the collection of the player. Note that a second copy of a card should not be allowed to be added to the deck. The collector should be notified if this is the case.
   d. **View Deck** – display as a list all the cards that are in the deck. The collector should be given the option to view a card in the deck by selecting either its name or its number in the deck. Doing so will display all the details of the selected card.

Here are a few clarifications to help better understand the limitations/scope of the specifications:

- The name of the card is its unique identifier. While you can have any number of copies of a card with the same name, their details must all be the same. This includes the card's variant. This means, if a card called "Pikachu" is added to the collection whose variant is "alt-art", no more other cards with the name "Pikachu" may be added unless it is a second copy of the same "alt-art" "Pikachu".
- The collection of a collector has no limit. You can store as many cards as you want in the collection, and any number of copies of a card can be stored (limited practically by the max integer size of course).
- A binder can only have twenty cards, but can hold any number of copies of a specific card (limited of course by the size of the binder).
- A deck can only have ten unique cards (i.e. cards with different names).
- A collector cannot trade cards directly from the collection. Cards for trade must be first put into a binder. As such, the trade functionality is connected to the binder (but may not be a method of the binder, depending on how you design your TCIS), not the collection.

Other Requirements

1. The design and implementation of the solution should…
   - Conforms to the specifications described above
   - Exhibit proper object-based concepts, like encapsulation and information-hiding
2. Interaction with the user (i.e. input and output) should be through a command line interface (CLI). No graphical interface is expected for MCO1.
3. To allow for an easier time to validate the program, usage of libraries outside of what is available in the Java API is not allowed.

Please also note that MCO2 will look to extend some mechanisms of MCO1; however, MCO2's specifications will be delayed in order to simulate additional mechanics or modifications requested by a client after development of the base system. Additionally, a graphical user interface should be implemented only for MCO2.

Screenflow and Text-based User Interface

As the application loads, show the user a menu that allows them to select what action they would wish to perform. Initially it would have the following options, but more options can be added as binders and decks get added to the collection:

- Add a Card
- Create a new Binder
- Create a new Deck

Once a card exists in the collection of the user, a new option should now be added to the main menu that would allow the user to increase or decrease card count.

Binders and decks can be created even without cards in the collection. Of course, if this is the case, the binders and decks cannot be populated with anything. Upon creation of a deck or binder, change the menu options from Create a new Binder to Manage Binders, and from Create a new Deck to Manage Decks. Selecting these menu options takes the user to the menu options found in the **Manage Binders** and **Manage Decks** sections respectively.

When going through any menu item, add a Go back to Main Menu option when needed. This should bring the user back to the Main Menu, cancelling whatever they were currently doing (e.g. as the user is inputting details about a card, they decide they don't want to proceed anymore; in this case, the card they were inputting details for won't be added to the collection).

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
    - Kindly ensure that the diagram is easy to read and well structured
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
5. A video demonstration of your program
    - While groups have the freedom to conduct their demonstration, a demo script will be provided closer to the due date to help with showing the expected functionalities.
    - The demonstration should also quickly explain key aspects of the program's design found in the group's class diagram
    - Please keep the demo as concise as possible and refrain from adding unnecessary information

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 the appendix.

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[1]. Comply with the policies on collaboration and AI usage as discussed in the course syllabus.

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 appropriate command-based instructions to create the documentation, but it must be PROPERLY constructed.

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
No bonus points will be awarded for MCO1. Bonus points will only be awarded for MCO2. To encourage the usage of version control, please note that a small portion of the bonus points for MCO2 will be the usage of version control. Please consider using version control as early as MCO1 to help with collaborating within the group.

Resources and Citations
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 Delivery
The mode of delivery of the demo for MCO1 is left to the discretion of your instructor. All members are expected to be present in the video demonstration and should have relatively equal parts in terms of the discussion. Any student who is not present during the demo will receive a zero for the 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 (for MCO1) and Object-Oriented (for MCO2) Programming concepts properly. No brute force solution. When in doubt, consult with your instructor.

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.

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

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 ID number	Student 2 Name 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

Class: MyClass						
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. **20**	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. **15**	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. **10**	The UML class diagram does not include most information based on the specs. **5**	No submitted UML class diagram. OR Design of classes are not in compliance to a logical object-based design. **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	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	No submitted class diagram to compare with.

	10	necessary features are still provided. **8**	**5**	features are still apparent. **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. **10**	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. **8**	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. **5**	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. **3**	No internal documentation. **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**
Delivery and Presentation	All members are present in the video presentation and have relatively equal parts to present. All members exhibit mastery of their project throughout the demo. **10**	All members are present in the video presentation and have relatively equal parts to present. All members exhibit familiarity of their project throughout the demo. **8**	All members are present in the video presentation; however, some members have little contribution to the demo. Most members exhibit familiarity of their project throughout the demo. **5**	All members are present in the video presentation; however, some members have little contribution to the demo. Most members exhibit some knowledge of their project throughout the demo. **3**	Did not appear in the demo. OR Member did not exhibit ample knowledge about the design AND implementation of the project. **0**