



De La Salle University
College of Computer Studies
Software Technology Department

CCPROG3 Major Course Output Program Specifications

Digital Calendar Application

Release date: May 28, 2025

Major Course Output Part 1 - due **June 28, 2025 (9PM)**

Your task for MCO1 is to create your own implementation of a Digital Calendar application. One of the available digital calendar applications you might be familiar with is Google Calendar. As creating an actual Digital Calendar system requires a huge team and longer development time with all the ideal features it can offer, the requirements for this project is narrowed down as below:

1. Account Creation / Deletion

At the start of the application, users will be asked to login in with their Account credentials: *username* and *password*. Since this is the first page, it is expected that there's no available Account to be used yet. Thus, there should be an option for the user to create an Account by providing their chosen username and password. Note that the username should be unique, no two Accounts can have the same username. The username will always be displayed in all pages once logged in.

The application should offer an option to logout from the Account anytime and login using another Account. An Account can also be "deleted" (meaning it cannot be used to login anymore but is still in memory) by the currently logged in Account by updating its active attribute from true to false (boolean type). Note that deleting an Account only deletes the private Calendars created by it leaving the public Calendars.

2. Add Calendar

There can be multiple Calendars linked with an account. The application should always display a list of the Calendar names linked to the currently logged in Account. By default, an Account has a private Calendar with the same name as the Account's username.

If a user decides to add a new Calendar to the Account, the user has an option to choose from the list of publicly available Calendars or create an entirely new one and decide the availability - private or public. Making a Calendar private means that only the Account who created it can see it; but if it is set to public, then anyone trying to add a Calendar to their Account can see the Calendar in their list and choose to add it.

3. Monthly View

The application centers on displaying a text-based monthly calendar together with the username, option to logout, and list of Calendar displays. The user-friendliness and ease of use of the overall **text-based user interface** is part of the assessment.

Upon logging in, the month to be displayed first should be the actual (real-world) current month in the current year. Users can move to the next or previous month and also have an option to jump into any month in any year. All Entries of the Calendars added to the Account should also be displayed in its exact date. Only the title of the Entry is visible. For dates that have multiple Entries, display should be sorted by the start time. A date has to be selected to see complete listed and properly formatted information of each Entries.

4. Add/Edit/Delete Calendar Entry

Each Calendar can have zero to any number of Entries. An entry is composed of the date (*required*), title (*required*), time start (*required*), time end (*required*), and the details. An Account can add, edit, and delete any Entry associated to all its Calendar - regardless if private or public. Once an Entry has been deleted, it will be removed from the memory.

5. Delete Calendar

Only the account that created the Calendar can have an option to delete it but an Account's Calendar named after its username cannot be deleted. It will only be deleted once the Account was deleted. Deleting a calendar should delete (in the memory) all the Entries created on it. The deleted Calendar should also be removed from the other Accounts that added the Calendar to their list and will not be visible anymore.

Implementation Requirements

The project must abide by the following implementation guidelines:

- The program should be fully implemented in the **Java** programming language.
- Observe the proper use of programming constructs (e.g., conditional, loops, data structures, classes).
 - Do not use brute **force**.
 - Codes must be **modular** (i.e., split into several logical methods/classes). Codes that are not modularized properly will not be accepted, even if the program works properly.
 - Exhibit **proper object-based concepts**, like encapsulation and information-hiding
- You can use these in the future when you have much more experience and wisdom in programming.

While you are in the CCPROG series you are NOT ALLOWED to do the following:

- to declare and use **global** variables (i.e., variables declared outside any classes)
- to use **exit**, **goto** (i.e., to jump from code segments to code segments), **break** (except in switch statements), or **continue** statements
- to use **return** statements to prematurely terminate a loop, function, or program
- to use return statements in **void** functions
- to call the **main()** function to repeat the process instead of using loops
- You may only use what is available in the Java API.
- You may use topics outside the scope of CCPROG3, but this will be self-study.
- Adhere to coding conventions:
 - **Class names** start with an uppercase
 - **Methods** and **variables** should start with a lowercase
 - Apply snake case for **constants** (in all uppercase) e.g. SNAKE_CASE and camel case e.g. camelCase for everything else
 - Proper indentation
- Debug and test your program exhaustively
 - The submitted program is expected to compile successfully and run in command prompt. 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.
 - As part of the requirements, you are required to create a test plan and execute your program against it. See [Appendix C. Example of Test Script Format](#)
- Since you will be working in pairs, you have to use code versioning for better access and collaboration. You should use **GitHub** service and follow instructions from [Appendix A. Git and GitHub Setup](#). Note that only pulling copy and pushing changes using **git** will be explained, other functionalities such as branching will not be covered. You may use just the main branch in working together with your pair. It is expected that both of you will push updates **at least once a week**.

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 B. Template for Declaration of Original Work](#) 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. Zipped **Javadoc**-generated documentation for proponent-defined classes with pertinent information
4. **Source code files** with proper internal documentation (**DO NOT ZIP**)
5. **Test plan** following the format indicated in [Appendix C. Example of Test Script Format](#)
6. A **video** demonstration of your program (**only for MCO1**)
 - Record the video demo using Zoom (no cutting / editing allowed). Both students should show themselves on camera and each have time to explain while sharing screen. Any student who is not present during the demo will receive a zero for the phase.
 - A demo script to be followed for the presentation will be provided closer to the due date to help with showing the expected functionalities.
 - The demonstration should **FIRST** quickly explain key aspects of the program's design found in the group's class diagram - especially the class relationships
 - Keep demo as concise as possible (5-10mins), refrain from adding unnecessary information.
 - **No submitted demo means 0 for MCO1 even if other files are submitted**

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.

Grading

For grading of the MCO, you may refer to the rubrics in the syllabus or in [Appendix D. Rubrics for MCO1](#) and [Appendix E. Rubric for MCO2](#)

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

Bonus Points

No bonus points will be awarded for MCO1. Bonus at most 10 points will only be awarded for MCO2 only with the professor's discretion. 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.

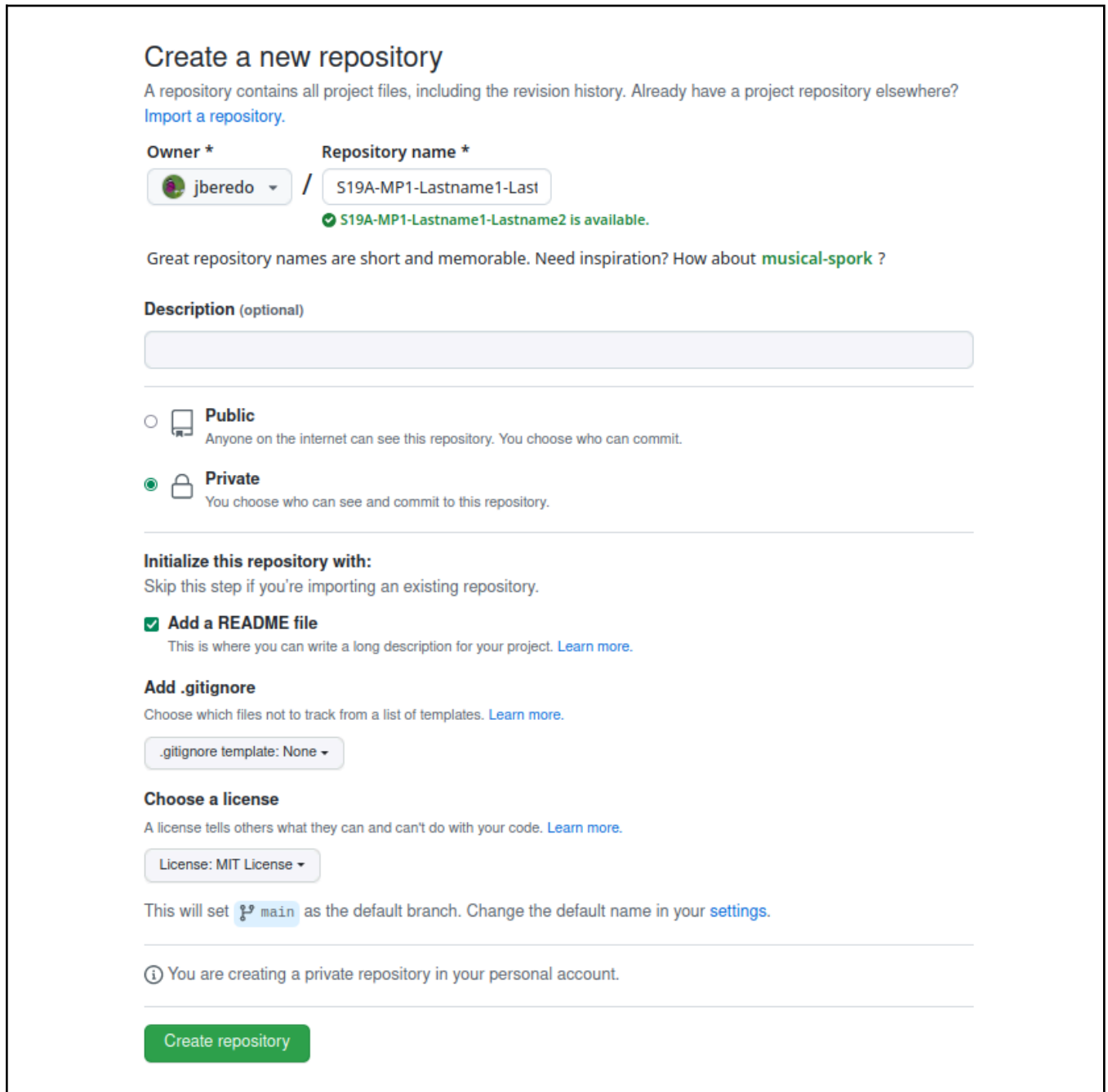
Resources and Citations

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.

For sources coming from internet videos or sites, provide the name of website, date accessed, and link.

Appendix A. Git and GitHub Setup (per Pair)

1. Install git on your local machine by following the detailed tutorial depending on your operating system in this link -> <https://www.linode.com/docs/guides/how-to-install-git-on-linux-mac-and-windows/>
2. Create an account on github.com using your DLSU email.
3. From the same site, create a 'New' repository and follow the input as shown in Figure 1.
 - a. Repository name should be your **Section-MP#-Lastname1-Lastname2**
 - b. Repository should be set to Private
 - c. Tick the 'Add a README file' option
 - d. Click 'Create Repository' button
 - e. Your repository should contain a LICENSE and README.md files



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * jberedo / Repository name * S19A-MP1-Lastname1-Last

✔ S19A-MP1-Lastname1-Lastname2 is available.

Great repository names are short and memorable. Need inspiration? How about [musical-spork](#) ?

Description (optional)

☐ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☒ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: None

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

License: MIT License

This will set main as the default branch. Change the default name in your [settings](#).

☐ You are creating a private repository in your personal account.

[Create repository](#)

Figure 1. Instructions for Step 3

4. Once done, add your partner as a collaborator on the created repository.
 - a. In your GitHub repository, click Settings then under 'Access' click 'Collaborators'.
 - b. In 'Manage Access', click 'Add people' and enter the email address of your partner.
 - c. Choose the email suggestion then click 'Add .. to this repository'.
5. In your command prompt, git may ask you to set your GitHub username and email. To do this, follow the git commands below:
 - a. `git config --global user.name "<username>"`
 - b. `git config --global user.email "<email>"`
6. Clone the repository to your machine using the command line.

NOTE: Repository name in the succeeding figures is *S15ABeredoJ*

 - a. Open your newly created repository, click 'Code' and choose 'HTTPS' from the mini tab, then click the copy button beside the link (**Figure 2**).
 - b. On your command prompt, change the directory (folder) to the location where you want the cloned directory to be saved (**Figure 3**).
 - c. Type **git clone** in the command prompt, paste the link you copied then press 'Enter'.
 - `git clone https://github.com/jheiy/S15ABeredoJ.git`
 - d. Enter your GitHub username and password. For the password, you have to create a token first.
 - Click the dropdown option with your profile picture then choose **Settings**.
 - Go to **Developer settings** then to **Personal access tokens -> Tokens (classic)**
 - Click '**Generate new token (classic)**'
 - For the 'Note' field, just type in anything eg. *CCPROG3*
 - In Expiration, choose 'Custom' then set until August 30.
 - Tick all the checkboxes.
 - Then 'Generate token'.
 - Make sure to **copy and save** to a file the generated token displayed with a light green background as it will not be visible after. You will use this every time you update the repository in place of a password.

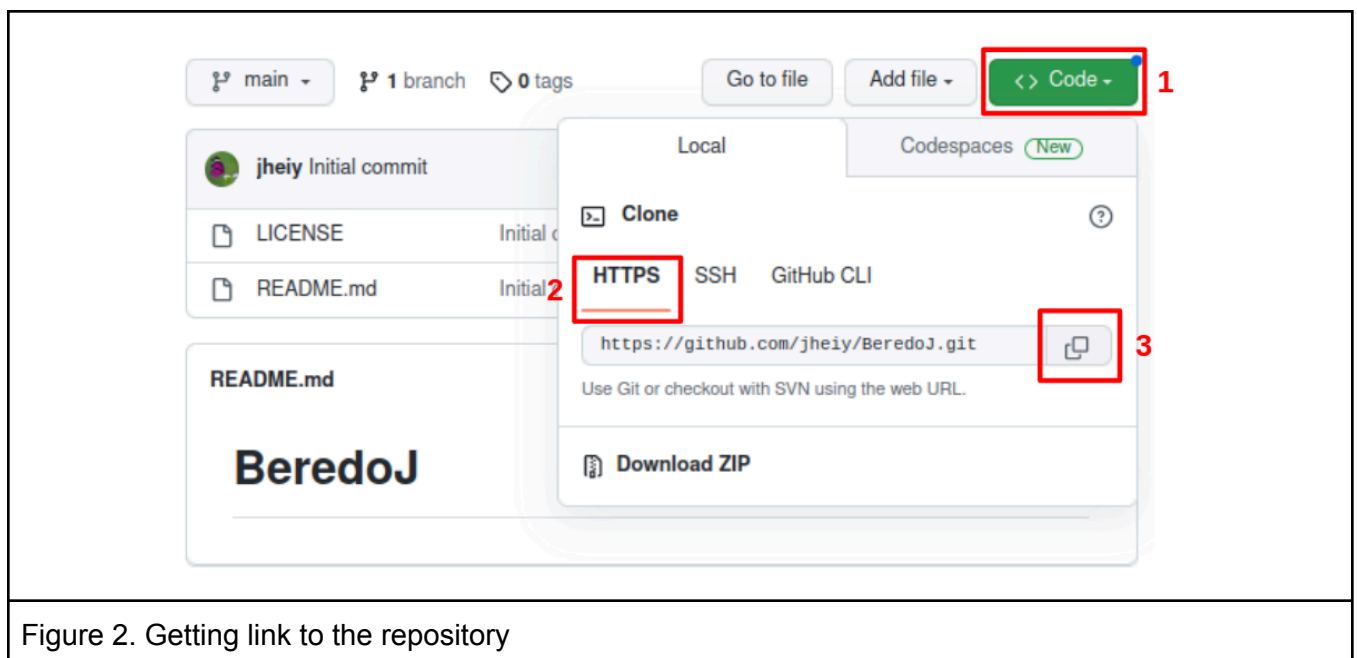
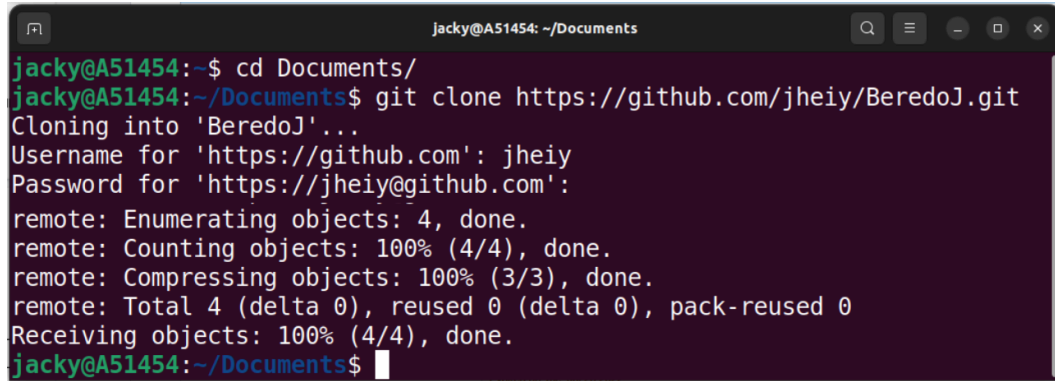


Figure 2. Getting link to the repository

A terminal window titled 'jacky@A51454: ~/Documents' with a dark background and light text. The user enters 'cd Documents/' and then 'git clone https://github.com/jheiy/BeredoJ.git'. The terminal shows the progress of cloning, including enumerating, counting, and compressing objects, and finally receiving all objects. The prompt returns to 'jacky@A51454:~/Documents\$' after the clone is complete.

```
jacky@A51454:~$ cd Documents/
jacky@A51454:~/Documents$ git clone https://github.com/jheiy/BeredoJ.git
Cloning into 'BeredoJ'...
Username for 'https://github.com': jheiy
Password for 'https://jheiy@github.com':
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
jacky@A51454:~/Documents$
```

Figure 3. Cloning repository

7. In your computer, check that a folder of your repository name (Section-MP#-Lastname1-Lastname2) is created.
8. Open that folder and create your first Java file.
9. Push your newly created (file) changes to your GitHub repository.
 - a. It is recommended to push changes every time you complete an additional feature no matter how small it is.
 - b. You may use the milestone statements provided at the end of this file as your commit message.
10. Here are some useful git commands:
 - a. [to see changes/updates in your local repository] **git status**
 - b. [to add an updated file to the GitHub repository] **git add <filename>**
** You may type git status again to see that there are 'Changes' now to be committed to the GitHub repository*
 - c. [to add all updated files to the GitHub repository] **git add ***
 - d. [to commit changes to GitHub repository] **git commit -m "<commit message>"**
** Commit message should be a short description of your update eg. "Added password feature"*
 - e. [to push changes to GitHub repository] **git push**
** DO NOT forget to push your changes after every commit to sync your local and GitHub repository copies*
** Username and password (token) will be asked in this command again*
** You may now refresh your GitHub repository to see the changes.*
11. Make sure to also add me as a collaborator in your repository.
 - a. EMAIL: **jackylyn.beredo@gmail.com** NOT dlsu.edu.ph
 - b. **DEADLINE** to share is **June 6, 2025 11:59PM**

Appendix B. 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

Student 1 Name
ID number

Signature and date

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 C. Example of Test Script Format

MyClass						
Method	#	Test Description	Test Input	Expected Output	Actual Output	P/F
getAverage	1	arr contains only non-negative numbers	arr = {1,2,3,4,5,0} arrSize = 6	2.5		
	2	arr contains only negative numbers	arr = {-1,-2,-3,-4,-5} arrSize = 5	0.0		
	3	arr contains a mix of positive numbers, negative numbers, and zeros	arr = {-1,0,5,-4,7,1} arrSize = 6	3.25		
	4	arr contains only zeros	arr = {0, 0, 0} arrSize = 3	0.0		
anotherMethod	1		
	2		
...		

Given the sample method above, the following are 4 distinct classes of tests:

1. Testing with arr containing only non-negative numbers
 - Non-negative numbers (positive numbers and zeros) are the only values that can be part of the average computation
2. Testing with arr containing only negative numbers
3. Testing with arr containing a mix of positive numbers, negative numbers, and zeros.
4. Testing with arr containing only zeros
 - Zero is a non-negative number

The following test descriptions are **incorrectly** formed:

- **Too specific:** Testing with arr = {1, 2, 3, 4, 5}
- **Too general:** Testing if the function can correctly compute the average of the non-negative numbers

Not necessary (because already defined in the pre-condition): Testing with a negative arrSize

Appendix D. 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	There are minor inconsistencies in design or implementation of	There are minor inconsistencies in design or implementation of attributes	There are significant inconsistencies between design and	No submitted class diagram to compare with.

	the presented Object-based design. 10	attributes or methods, but all necessary features are still provided. 8	or methods, which leads to missing or unexpected features. 5	implementation at the class level, but most 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

Appendix E. 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 is following Model-View Controller. 15	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 Design is following Model-View Controller. 10	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 Design is not following Model-View Controller. 6	The design provides some of the original specs and includes unnecessary or redundant classes. AND Design is not following Model-View Controller. 3	No submitted UML class diagram. OR Design of classes are not in compliance to a logical object-oriented design. 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 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, 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. 3	No internal documentation. 0

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