# SOLID

We think our project is consistent with the SOLID design principles. We will explain why it is the case from five different perspectives.

## Single responsibility principle (SRP)

We think our project follows SRP because every class created by us is responsible to one, and only one actor. Initially for each of the classes we have created such as User, Resource and Achievement, we have created managers for each of them. It contains very little code but is responsible for instantiating and delegating to the classes with the functions add, delete and modify.

In addition, our previous design of Presenter did not satisfy this principle, it had its own scanner and scanning inputs on its own. In this case the presenter was doing the controller’s job and failed to have only one responsibility. We solved this problem by adhering to the single responsibility principle by moving all the scanner parts to the controller SystemInOut and then to CommandLineInput so that the Presenter has only one responsibility: present. All the input jobs are deferred to UI and controller where the UI reads input strings to the controller which delegates to different Use Cases. By doing so, we adhered to the Single Responsibility Principle.

## Open-Closed Principle (OCP)

We think our project follows OCP because we have the class that opens for extension but it is not dependent on another class that is closed for modification. The class GraphBuilder is the class that is open for extension. In our project, the classes IntroCSGraphBuilder, MathematicsGraphBuilder and IntroMakeupGraphBuilder extend GraphBuilder because they share the functionalities that are related to buildDirectedEdges and buildGraph. However, they are different in terms of how they are initialised. In this case, we make a class GraphArchitect that is closed for modification. It could build graphs of all the subclasses of GraphBuilder and we could add more GraphBuilder without changing anything to the class Publish. Therefore our project follows OCP.

## Liskov Substitution Principle (LSP)

We think our project follows LSP because we have created many interfaces and superclasses that are implemented and extended by subtypes. One of the examples would be PublishedContent. In our project, the classes Post and Comment extends PublishedContent. Whatever the method that needs a PublishedContent variable, it could be replaced by its subclass Post and Comment without any error.

Our UIAdapter Interface also satisfies LSP. We defined a UIAdapter Interface with different UIs that implements it. The controller takes in a UIAdapter and thus depends on abstraction instead of UI. In our main program, we can freely instantiate different Uis and pass it to the controller and execute the program. Our program will work perfectly fine on different UIs. In this case, different UIs of our program can substitute the UIAdapter to be passed into the controller.

## Interface Segregation Principle (ISP)

We think our project follows ISP because we try to keep our interface as small and specific as possible so that no class implements the method that is unnecessary. For example, instead of making one big interface with all functionality of PublishedContent, we have broken it down into Likable, Visible and Serializable and each interface is in charge of different methods.

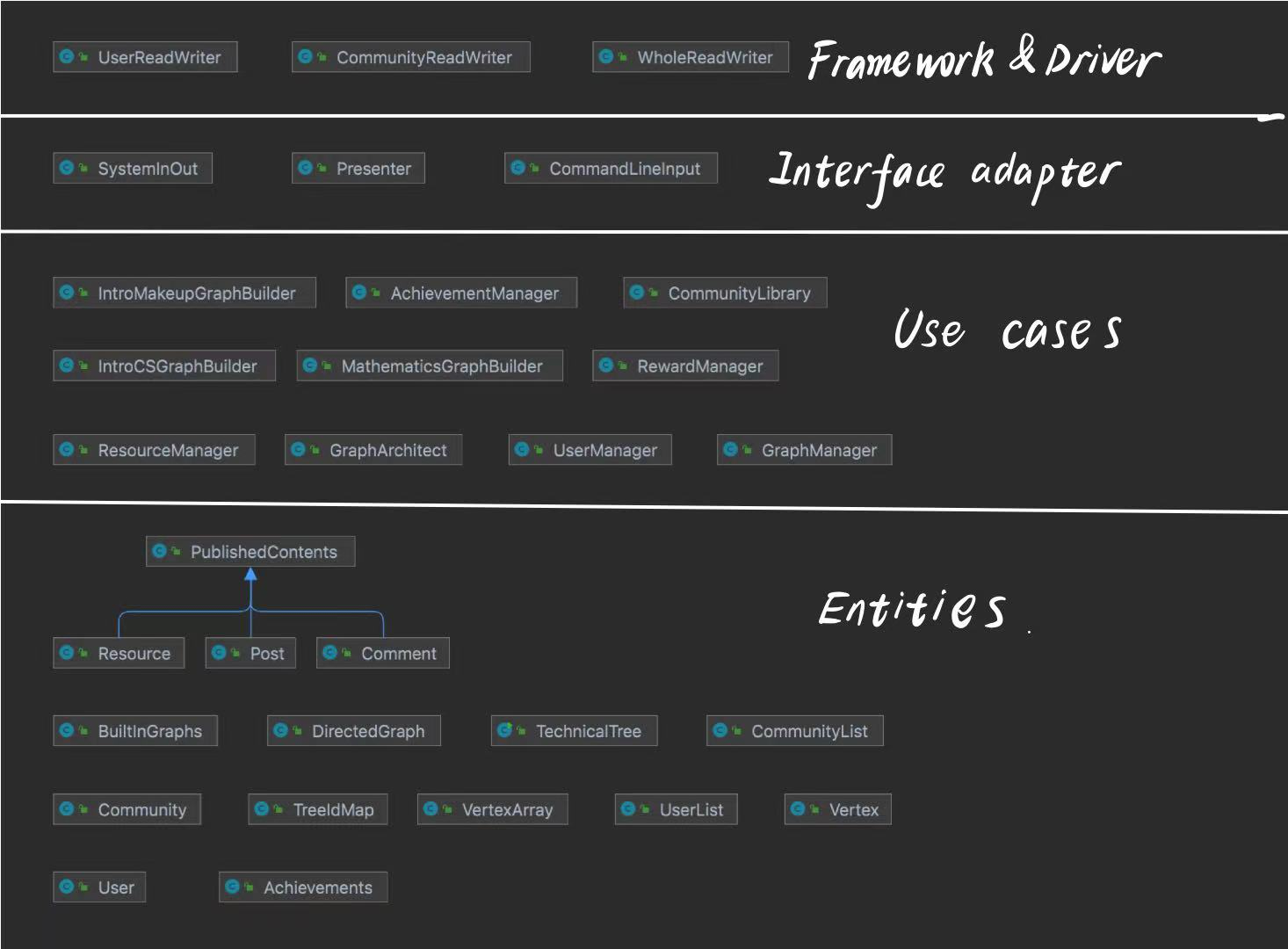
## Dependency Inversion Principle (DIP)

We think our project follows DIP because we have a class that has a variable type interface. In our project, our class GraphArchitect has only one variable graphBuilder of type interface GraphBuilder. In addition, our controller also does not depend on a specific UI. Before, our controller SystemInOut had a scanner as an instance variable and scan command line inputs to interact with the user. In this case, the controller was dependent on the UI and our program was thus fixed to the command line UI. However, we created a UIAdapter Interface which specifies a method getInput which is implemented by UI and our controller depends on it. The UI CommandLineInput implements it using scanner for the getInput method and that is now how our program interacts with command line user inputs, but our controller SystemInOut takes a UIAdapter UIAdapter as an instance variable and call its getInput method whenever needed. In this case, the controller depends on abstraction and is no longer hard coded to command line inputs.

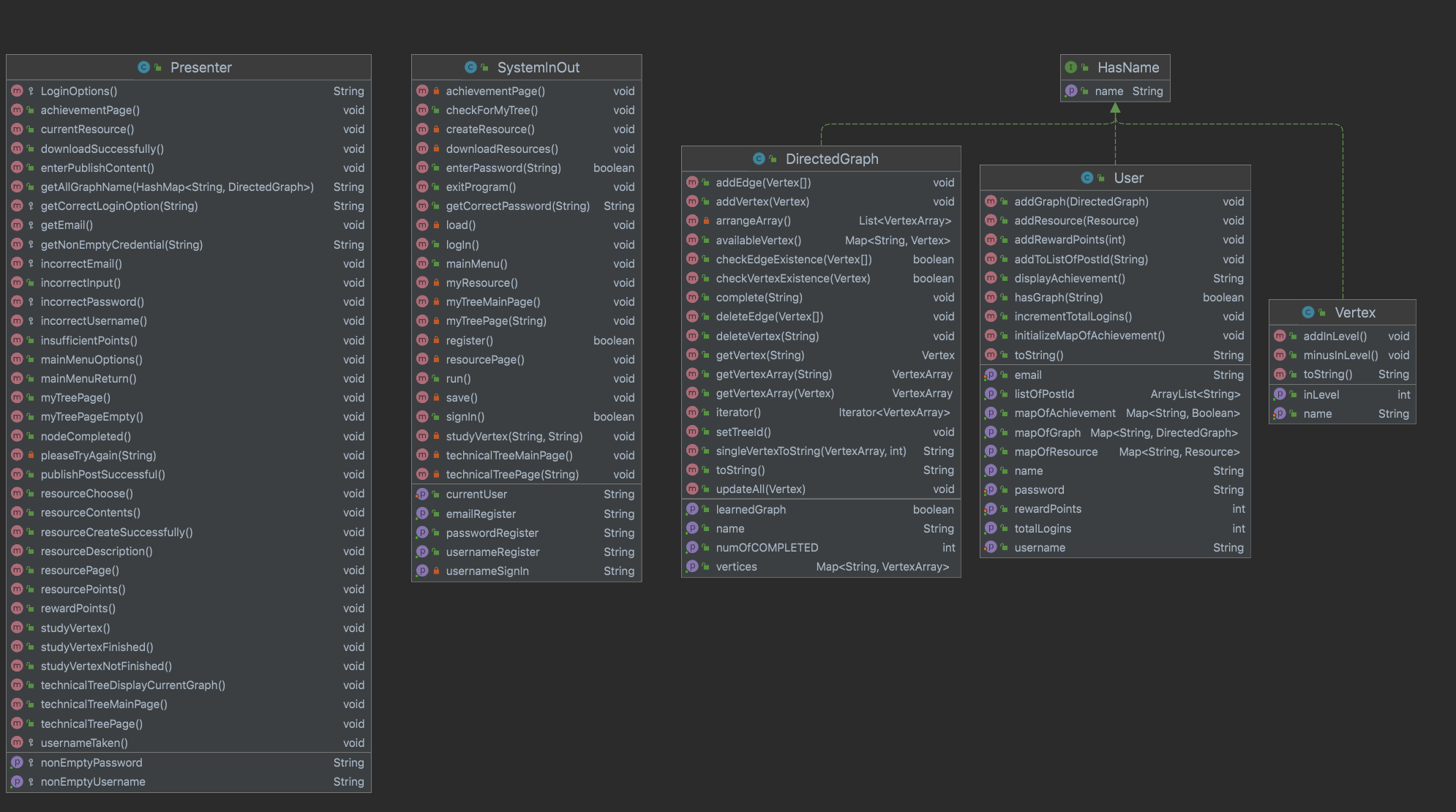
# Clean Architecture/Class Diagram

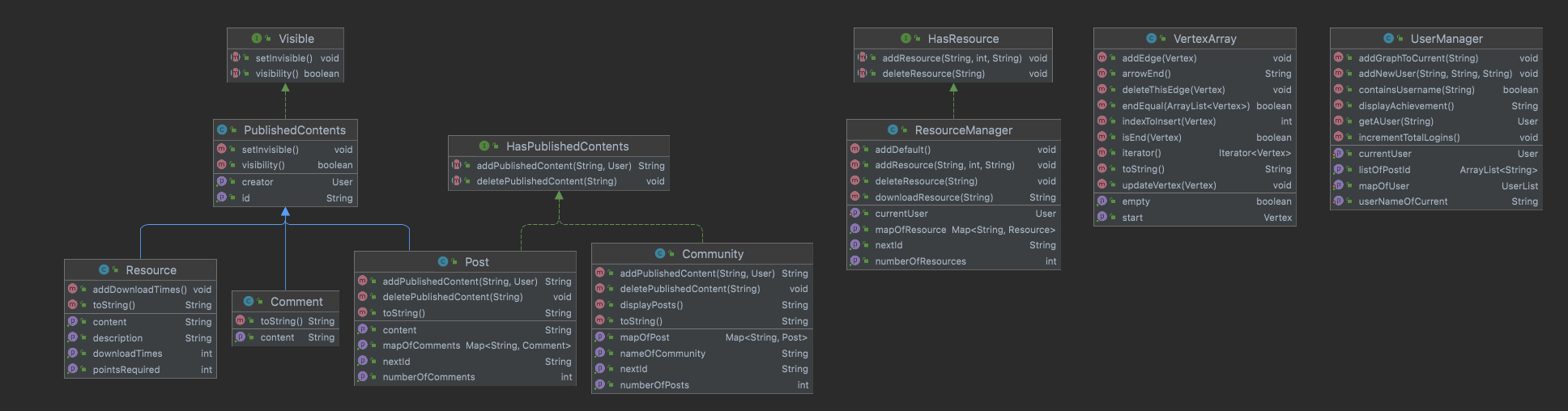
## UML diagram

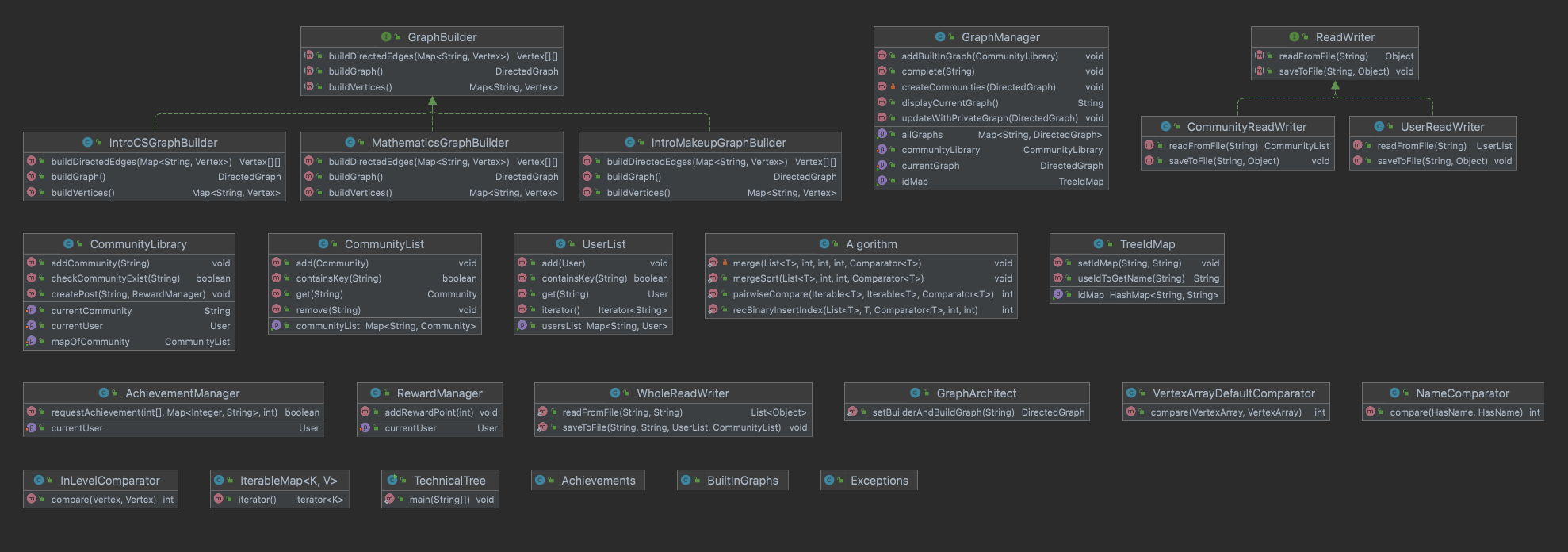
(For simplicity I omitted the variables and the methods at here, but I attached the complete UML graph in the last)



## Complete Version of Class Diagram:







## Scenario Walkthrough:

I will use a short scenario walkthrough to illustrate that our program here adheres to clean architecture.

Suppose that a user is using the tech tree for learning purposes and he encounters some problems and now wants to put up a question post in the community to discuss with other people in the community. He would simply input the title of the post and the content of the post then press post. A post will then be created in the community. In this process, the program would execute in this way: First, our program will use CommunityManager, which is one of the use cases, to access the CommunityLibrary to find the intended community, this is the interaction between use cases. The CommunityLibrary would then find the specific community and access it. Now we have accessed an entity, a community, which follows the dependency rule. Then the add() method in the class community will take in the content and the User to create a post in this community. The interaction between community and User and community between posts are all interactions between entities. We haven’t violated any principle of clean architecture in this whole process.

Volations: Not any so far.

Is the Dependency Rule consistently followed when interacting with details in the outer layer?

* Give us a concrete example from something like your UI or an interaction with a database.

Given a scenario that a user now wants to login to his own account and this process involves the interaction between the database and our use case. I will demonstrate below that it follows the Dependency Rule:

When the account number and the password is read in from the user. Similar to the process of posting a post in the community. The systemInOut class, which is the presenter class, will take in the credentials input by the user and pass it on to the userManager, which is also a use case. As the userManager takes in the credentials, it will then compare the input credentials and verify whether it is a valid user. This is an interaction between use case and database. At last, if the credentials are correct, the userManager will find this specific user and access it. This is the interaction between use case and entity. In this whole process. The Dependency Rule is not violated.

Updated Clean Architecture:

In phase 0 and phase 1. We mainly worked on the entities and the use cases of our program, which is of course the core of the program. In phase 2, we mainly worked on implementing the systemInOut/presenter class and the serialization class. This allows us to focus on the interaction of our program with the user. During this process, we’ve also noticed the defection of our program and it gives us the chance to perfect it. Other than entities and use cases. The presenter and readWriter also added two more layers of Clean Architecture to our program -- Interface adapter and Framework&Drivers. I will use two small scenario walkthroughs to illustrate that we adhere to the dependency principle as we designed these two classes.

readWriter:

The main purpose of the readWriter class is to let the program be able to store data and access it in the future. Suppose a complete new user has opened up a brand new account by setting its own username and password. Serialization enables the user’s data to be stored and could be accessed in the future. This whole process involves the interaction between Framework&Driver, i.e. Serialization and interface adapter, i.e.presenter. This adheres to the dependency rule.

Presenter:

The presenter class plays the role of interacting with the user and the program itself. What we are seeing in the command line is presented by the presenter. Suppose that the user is prompted for entering the user name and the password to login. As long as the user has input the valid credentials. The user would be guided to the next stage. This process involves the interaction between use cases and presenter, possibly database and presenter if the user already has an account. This whole process adheres to the Clean architecture and dependency rule.

# Design patterns

|  |
| --- |
| Builder Design Pattern |
| Classes/Interfaces involved:  GraphArchitect (Class), GraphBuilder (Interface), IntroCSGraphBuilder (Class) |
| Using a builder design pattern to build built-in Technical Trees  GraphBuilder is the interface that declares the methods every Technical Tree builder should have. IntroCSGraphBuilder implements the GraphBuilder interface and specializes in building the Technical Tree, “CS Introductory Series”. GraphArchitect is a class that controls the Technical Tree builders. It receives a Technical Tree order, uses the corresponding Technical Tree builder to build the tree, then outputs it. |
| Pull request:  https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/1/files |
| Strategy Design Pattern |
| Classes/Interfaces involved:  InLevelComparator, NameComparator, VertexArrayDefaultComparator |
| Problem: When doing sorting and searching, every element needs to be comparable. However, for different data structures like Vertex, VertexArray, HasName, they need different strategies to compare and under different scenarios, the same data structure needs to be compared based on different ways. This requires the Strategy Design pattern. |
| In our program, we created several classes which all implement the Comparator interface and are used to compare data structures based on the requirement. InLevelComparator compares Vertices according to their inLevels, NameComparator compares two HasName’s according to their name. VertexArrayDefaultComparator compares two VertexArrays according to the inlevels of their ends. All these comparators will be used by sorting and searching algorithms by Dependency Injection which will be talked about in detail. |
| Pull Request:  <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/commit/413a51c75de3a4aded41373169685685f82ce567>  (Implementing different comparators following the Strategy Design Pattern |
| Dependency Injection Design Pattern |
| recBinaryInsertIndex, pairwiseCompare, mergeSort |
| Problem: sorting, searching, and comparing algorithms need a way to compare different data structures. Instead of hard-coding a comparing strategy, we want our methods to be opened to as many possible comparing strategies as possible. |
| We use the Dependency Injection Design Pattern. It is not the sorting, searching, or comparing algorithms’ responsibility to specify a comparing strategy, but it is codes that use these algorithms's responsibility to specify a comparing strategy and then tell the algorithms. So we have our algorithms consume a comparator argument which tells them how to compare, and use this comparing strategy to function. When these algorithms are called, we first initialize a comparator, and then pass it into the algorithm to proceed. |
| Pull Request:  <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/commit/413a51c75de3a4aded41373169685685f82ce567>  (Injecting comparator to sorting and searching algorithms instead of forcing them to decide the comparison strategy using comparable) |
| Iterable design pattern |
| IterableMap, DirectedGraph, VertexArray |
| Problem: In many cases, we need a way to independently loop over elements stored in different data structures, fetching each element one at a time. We thus need the Iterable Design Pattern |
| To access the benefit of constant time inserting, deleting, and retrieving data from a collection, we extensively use Map abstract data type throughout our program. However, Map is not Iterable. In many cases, we need to perform an action on every element in the Map, so we designed IterableMap which is a subclass of HashMap which supports looping over all its keys. Also we make DirectedGraph and VertexArray iterable, supporting iteration over them. When iterating over DirectedGraph, it will fetch its stored VertexArray one at a time for use. When iterating over VertexArrays, it will fetch its stored ends of every directed-edges for use. |
| Pull Request:  <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/commit/eabbefd3ba6d8a653e481b0df9abad016bf3426e>  <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/commit/712aa13fb0da86feed6dd5674b2de94b569d3edc>  <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/commit/fc23b5669949289f9337fc9da26ad9b47281f239> |

# Use of GitHub Features

## Branching

Every member has a own remote branch that they push to, and a local branch that they make changes in.

## Reviewing & Merging pull request

Whenever we complete writing our code, we use the Pull Request function to create a new pull request with descriptions of the changes done to the code and notify the other members for review. Then, other members of the groups will go over the changes that are made in the new pull request and comment on anything they think is good/need improvement. Once some members have reviewed and approved the pull request, we will merge the changes to the main branch and everyone will pull the new version of code.

## History

Since we use GitHub, it is easy for us to keep track of our previous version and get things back easily. If we submitted the wrong version and the whole project crashed, or some team member deleted some essential file by mistake, then we can get the history version and never lose our work.

## Issue

We used the Issues function to communicate some of the big issues we think need to be fixed in our code. We post issues on the board and treat the board as a discussion board, in order to keep track of the problem-solving process and/ or brainstorm process(we may discuss some new features for our program here)

In total, GitHub really helped us to effectively work as a collaborative group for the development of software projects. We can easily to git the latest version. And have different versions on our own laptop at the same time, we can resolve the conflict by comparing highlighted code and deciding the final version of the code as we wanted.

# 

# Code Style and Documentation

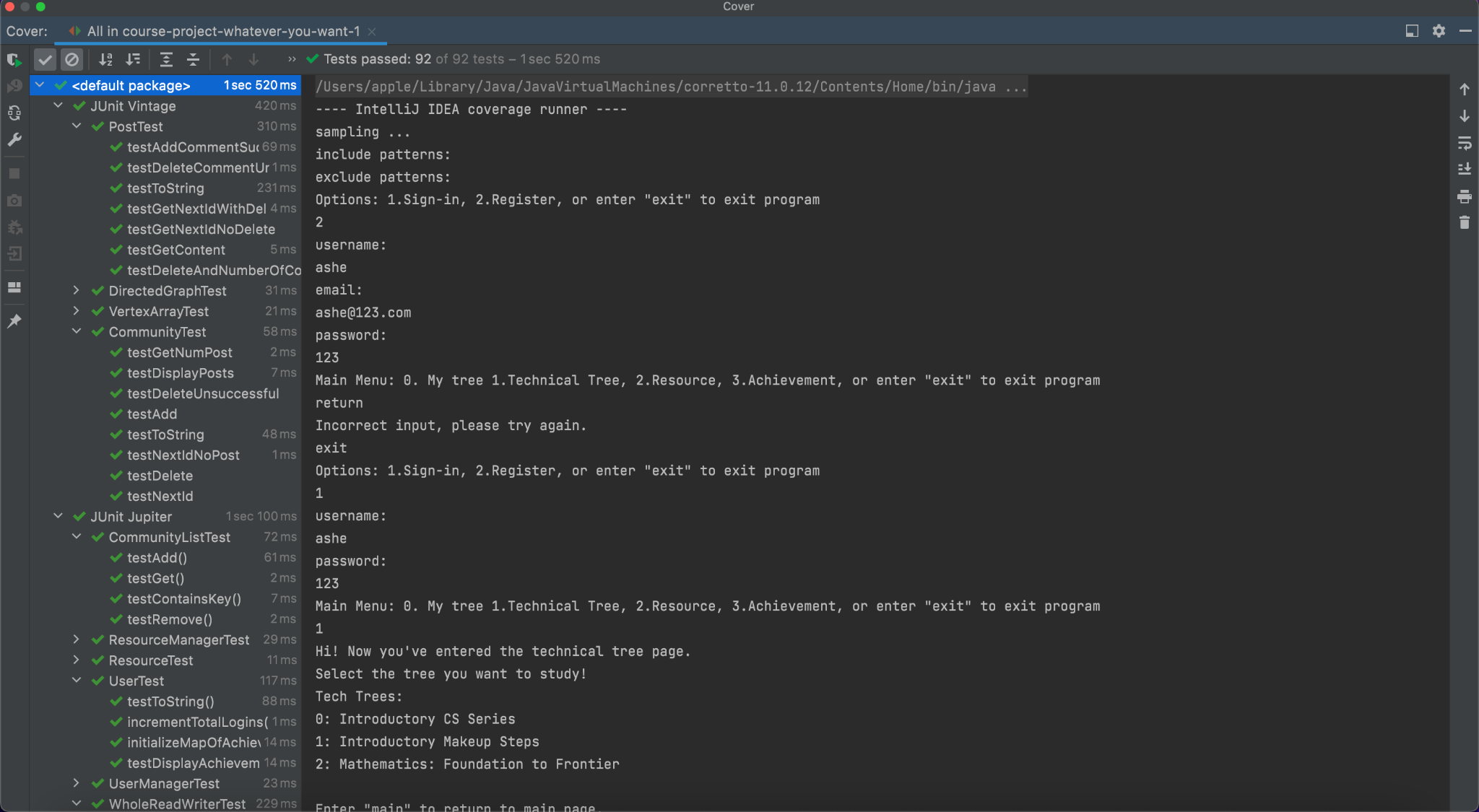
We included detailed descriptions of the changes we made to the code in our pull requests. We also reviewed and commented on the pull requests before merging them to the main branch.

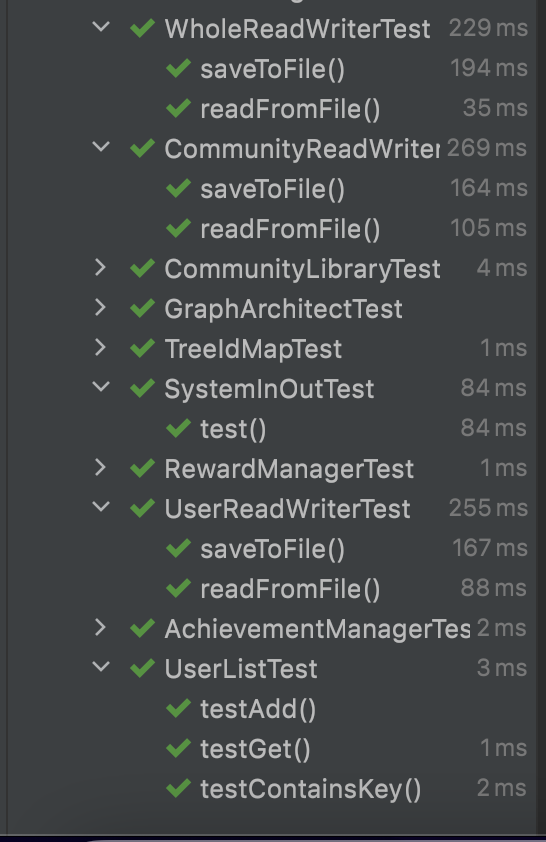
We do have some warnings in IntelliJ about unused variables and methods. These variables and methods are created for functions that we will extend in the future. These warnings will be solved once we start extending the functions in Phase 2.

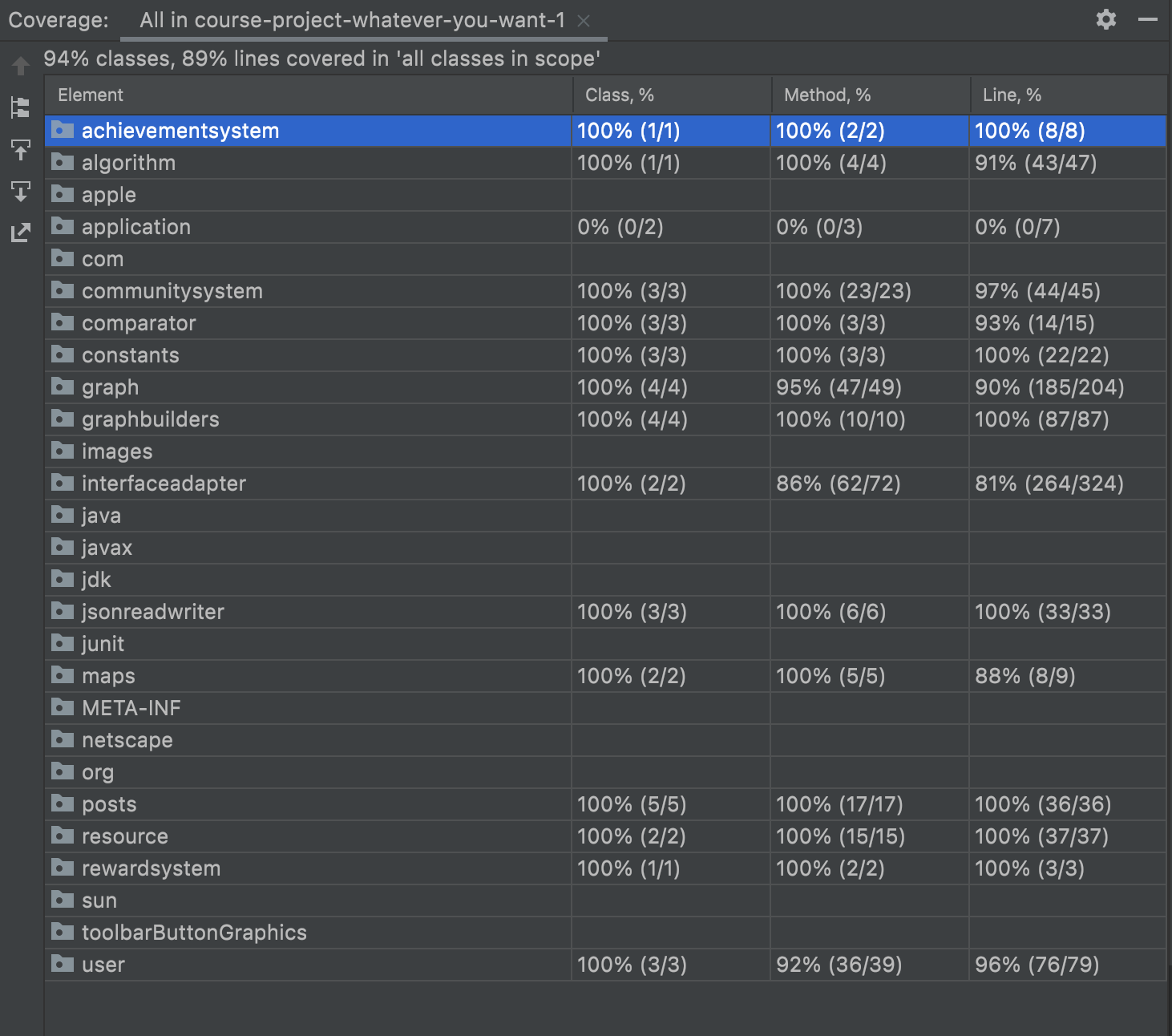
We also have full javadocs for our methods, especially for the more complicated ones, that explain what they are, what they take in, and what we should expect from them.

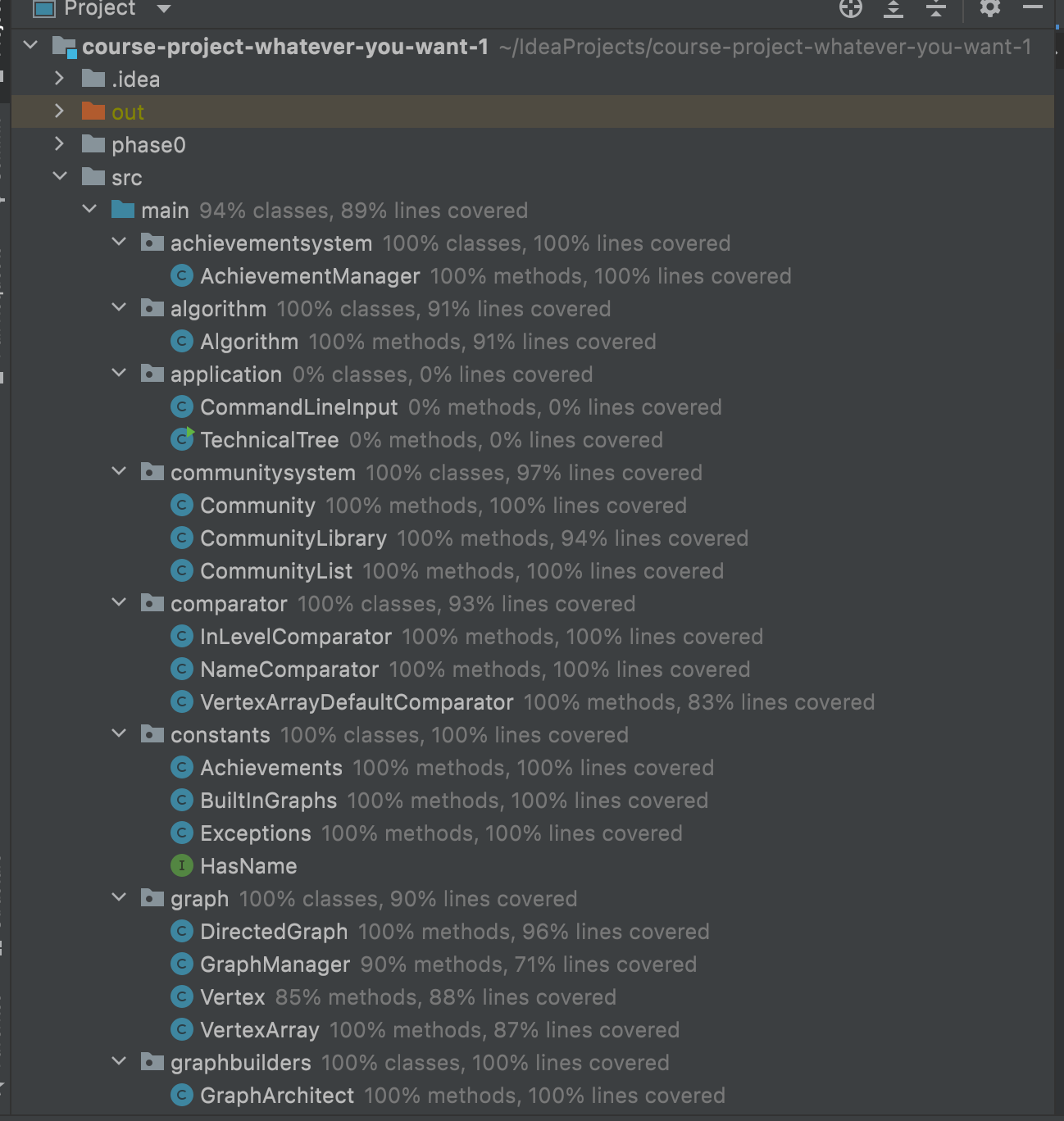
# Testing

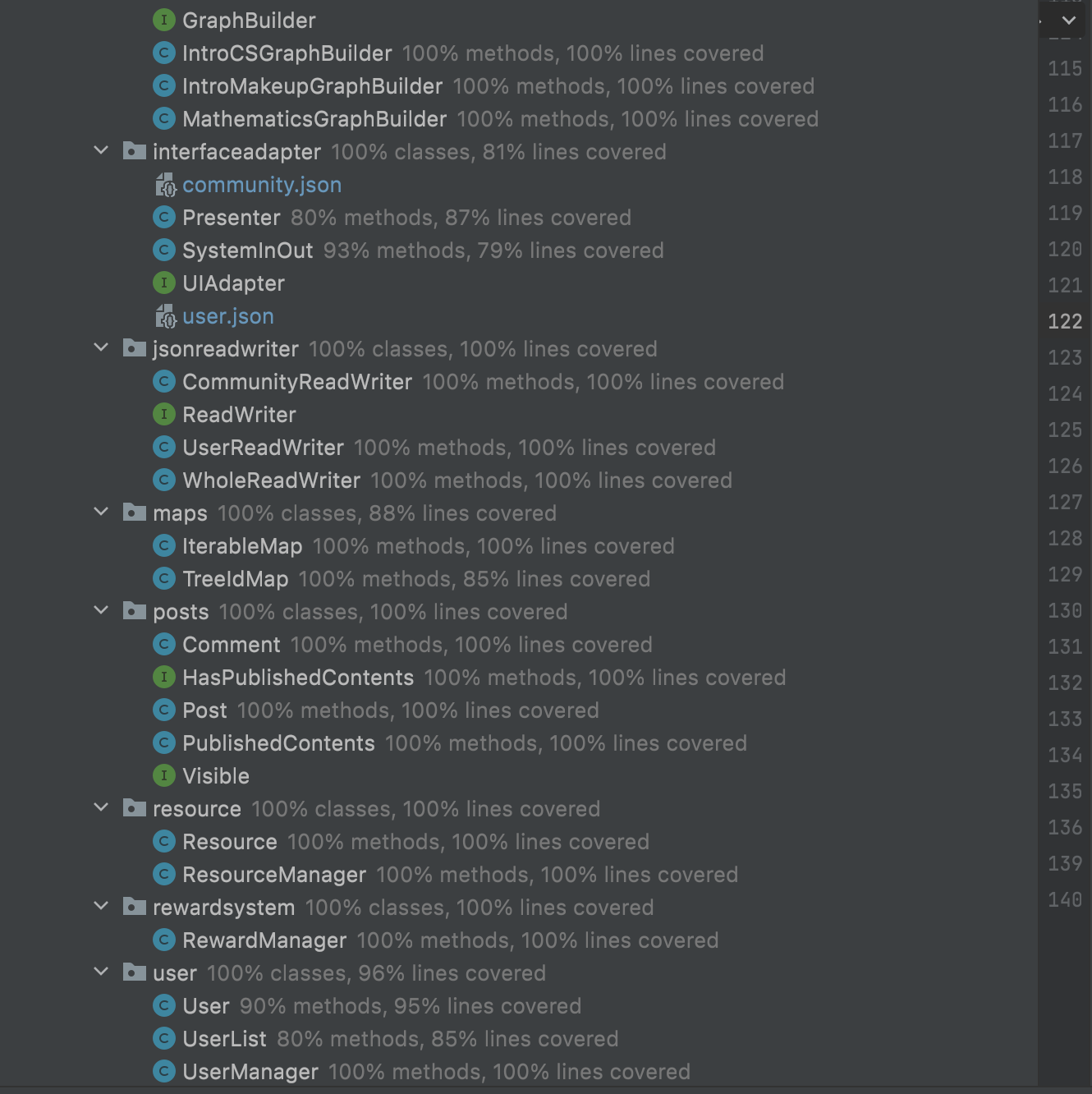
We have some tests that cover overall 94% of classes and 89% lines. The details of our tests are attached.











# Refactoring

We have used refactoring functions to rename class, variable or method names that are not clear enough. We have also used refactoring to create different packages for the classes and interfaces that share the same functionality.

We rearrange the packages so that all of the tests are in one package and the rest of code in another. We also renamed the packages so they follow the package naming convention.

# Packaging strategies

We considered packaging by component and by layer. We eventually settled with packaging by component because it allows us to quickly locate a class that we are looking for. For example, if we are looking for the class AchievementManager, we know that it is most likely to be in the package, AchievementSystem.

# Updated Specification

## Local Data Storage & Serialization

In Phase 2, data can now be stored locally using serialization. After users register their account, they can exit the program and sign back into their account later to restore their account information as well as their previous progress.

## Tree

In Phase 2, we updated the tree so that we have 2 new trees. We also updated the visualization of the trees so that they can be presented to the users in a more clear and organized way.

## SystemInOut & Presenter

In Phase 2, we added SystemInOut that receives user input and calls other classes accordingly. We also added a Presenter which contains all of the outputting strings related to user action suggestions and is called by SystemInOut

## UI

In Phase 2, we added UI for all of our program features that need to be visualized so information can be presented to users in a clear and organized manner. For example, we have an achievement page where users can see the current status of their achievements.

# Design decisions/Code Organization

We have two ways to implement user.achievementList. First one is to have a hashmap, and this hashmap has all achievements in our program, and the value for each one is True or False, which denotes that the user achieved or not achieved an achievement. For example, user.achievementList = {“achievement1”:True, “achievement2”:False}The other one is to have an arraylist for storing the achievement that the user achieved. For example, user.achievementList = {“achievement1”}. Both of them are describing the same thing. After discussing, we finally decided to use version1, because it would be easier for visualization of the achievement library of a user. The user will be able to look through all the achievements, no matter if they are achieved or not, so it will be easier to have True or False for each achievement if we want to implement the UI.

We need to have a resource class that initialises resources. However, how to download that resource becomes a problem. At first we thought that it was enough to download the resource inside the Resource class. However, we have encountered a problem that the resource could not be stored inside the user library. In this case, we have created a class called resourceManager that keeps track of all the resources. The resourceManager could not noly add or delete any resources but also download the resource and add one download times whenever the resource is downloaded. Since the resourceManager keeps track of all the resources, in future we want to add a search method that searches the resource that contains a specific keyword. We would also like to add a sort method that arranges the resource according to the download times.

The VERTICES instance variable of the DirectedGraph class used to be of type Map<String, Object[]> where the key is the reference (name) of a vertex and the value is all DirectedEdges that starts from the vertex referenced by the key, and Object[] is responsible for storing it. So previously, Object[] is an array of length 2. At index 0, it stores a vertex that is the starting vertex of all DirectedEdges Object[] stores. At index 1, it stores an ArrayList<Vertex> which stores all vertices that act as the ending vertex of DirectedEdges that start from the starting vertex. When we want to retrieve the ArrayList<Vertex>, Java actually thinks it as of type Object as it is stored in Object[], and when we want to apply any ArrayList methods on it, we need to cast it to type ArrayList<Vertex>, which cause an unchecked downcasting problem. To solve this problem, we implemented a new data structure called VertexArray to store all DirectedEdges with the same starting vertex. It has 2 instance variables: START which is the starting vertex of all DirectedEdges stores, and END which is an ArrayList<Vertex> that stores all vertices that act as the ending vertex of DirectedEdges that start from the starting vertex. Now, VERTICES in DirectedGraph is now of type Map<String, VertexArray>, and when VertexArray is retrieved, no casting is needed, which solves the unchecked downcasting problem.

# Progress report

|  |  |
| --- | --- |
| Code | |
| Alfred | 1. Implemented the framework of the new command line UI    1. It currently allows users to return to the main page..    2. Fully implemented the login and registration page.    3. Fully implemented the main page that displays the options to other sections.    4. Having the framework is important because it allows my teammates to have something to build on and it acts as an example that they can refer to. The “return” mechanism gives clients a lot of flexibility.    5. Please refer to the file SystemInOut.java in the below.    6. <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/63/files> 2. Implemented the part of Presenter for Login.    1. Now the responsibility of sending out prompts and receiving commands from the clients are separated into different classes.    2. The separation is important because it allows the extension of different modes of displaying texts to be fairly easy to implement. For example, if we want to change the language of our program to French, we only need to add a new presenter class, and none of the code in our controller class, SystemInOut, would be affected.    3. Please refer to the file Presenter.java in the link below    4. <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/94/files> |
| Coco | 1. Fixed all the tests so that they work with current modifications 2. Fully implemented the display of Post and Comment 3. Fully implemented the display of achievements and their corresponding status 4. Implemented tests for toString() for Post and Comment and tests for displayAchievement 5. Moved tests out of source file and reorganized the packages 6. Moved all outputting strings in SystemInOut into Presenter so that it follows SOLID principles 7. Completed the tests for all of the methods in RewardSystem and CommunityLibrary 8. Wrote the accessibility report   Pull request link:   1. <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/57>    1. Since we have made extensive amounts of modifications to our program since the date we wrote these tests, none of the tests actually test the current program. Because of all of the modifications, the tests all need to be rewritten which is quite workload heavy. However, it is important to keep the tests up to date with our program so that we know if the classes and methods are working correctly. 2. <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/77>    1. This is the pull request for the tests of the toString and displayAchievement that I implemented in separate pull requests (so I chose this pull request to demonstrate my contribution of all). It is important for the user to be able to clearly visualize the posts and comments and achievements since these are some of the main features of our program. It was also quite difficult to implement the toString of Post since it not only involves the Post itself but also all of the comments that come with it in an organized manner so quite a chunk of time was spent figuring that out. |
| David | 1. Create a VertexArray class to solve the unchecked downcasting problem. 2. Make DirectedGraph, IterableMap, and VertexArray iterable. 3. Include the all methods in Algorithm and multiple comparators and make VertexArray sorted in preparation for directedGraph’s toString() 4. Implement directedGraph’s toString() method with mathematics graph to test it 5. Implement SystemInOut Test and the UIAdapter Interface and the Command Line UI   Pull request link and comment:   1. <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/commit/03c797f8d06635e35362ee36687b807040ef5747>    1. In UI we always need a good way to visualize graphs and prerequisites of courses. Being the creator of DirectedGraph and VertexArray, I implement its toString to create better visualization. 2. <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/138>    1. Implement SystemInOut Test to test controller and presenter, so the test line coverage increased around 30% after this single commit    2. Adhere to Single Responsibility Principle so that Presenter no longer take inputs from UI but are all moved to controller    3. Apply Dependency Inversion Principle by introducing the UIAdapter Interface which UI implements and controller depends on so that the Controller no longer depends on UI directly but instead depends on abstraction. |
| Tong Su | 1. Refactored the name of the packages to lowercase 2. Changed the type declaration of HashMap to Map wherever appropriate 3. Write the command line method for downloadResource createResource and displayResource 4. Safely eliminate all the warnings of the project 5. Add some missing java doc 6. Remove the commented code 7. Complete resource related presenter and main menu return presenter   Pull request link for reference:   1. [.https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/120#issue-1071570414](https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/120#issue-1071570414)   This is important as by solving all the warnings in the project, it will make our project neat and appropriate for reading. It also ensures that our project functions well without any error.3.   1. <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/82#issue-1064734117>   Resource function is one of the important features in our project. By completing the resource part of the resource part of the command line, our users could proceed to the resource-related functions without any error. |
| Arthur | 1. Implemented parts of the command line UI 2. Fully implemented the display for tech tree 3. Fully implemented the study of nodes and posting comment after finishing the nodes 4. Added JavaDoc for some of the unfinished classes 5. Modified graphManager/achievementManager/ rewardManager/userManager to better cooperate with the systemInOut and the presenter class   Pull request link for reference:   1. <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/111> 2. <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/96>   Reasons for these two pull requests are important:  The systemInOut/presenter class plays the role of interacting with the user, therefore it is important to consider the needs of the user and also organize the display of our program. Connecting each class together to make it as a whole is the significance of the systemInOut/presenter class. I contributed to this part to make our program more complete :) |
| Ashley | 1. Implemented parts of the command line UI    1. Add the interface for My tree and store them properly    2. Change the way to judge the correct email address by regex    3. Change a few presenter method to made the code be reused easier in the future. 2. Serialization    1. Implemented the user readwriter and community readwriter    2. Implemented the whole readwriter    3. Made some class serializable    4. Fix some bugs    5. Edit SystemInOut file and decide where should the method should be called    6. Implement a explicit json readwriter before, but it is quite complex to further implement, so finally we choose the easier one 3. Make changes on User Class    1. Reorganized the structure for userinfo class(now it is changed to user class)    2. Add some attribute and methods for userinfo class 4. Wrote some javadoc and test.   Pull request link:  <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/113>  <https://github.com/CSC207-UofT/course-project-whatever-you-want-1/pull/86>  Reason for pick these two pull request:   1. The first one is almost the final version for our user interface. Fixes many bugs. 2. The second one created two new classes to help develop the clean architecture implementation and fixed bugs. |
|  | |

Accessibility Report

Principle 1: Equitable Use

* I think our project is equitable in use because we treat all users equally. This is shown in our registration process in which we do not ask for any user’s personal information such as age, gender, race, etc. Thus, we do not treat users differently based on their age, gender, race, etc .

Principle 2: Flexibility in Use

* I think our project is flexible in use because it is suitable for both left-handed and right-handed users. This is because our project is running using the command line of the IntelliJ. The user just types in the word in order to progress to the next step. In this case, regardless of whether the user is left or right handed, he or she could easily type out the command by using the keyboard without any difficulty.
* Since our program runs in the command line of IntelliJ, the users are free to adjust the appearance of our program by adjusting the setting of IntelliJ which allows them to choose font, font size, colour theme, etc.

Principle 3: Simple and Intuitive Use

* All of the possible actions the user can perform along with what they do are presented to the user at each stage of the program so that it is clear to the user what they can do and how to achieve their desired outcome.

Principle 4: Perceptible Information

* Since our program runs in the command line of IntelliJ, the users can adjust the font size of the program by adjusting the font size in IntelliJ. This allows users with visual limitations to adjust to bigger font size so they can better read our program.
* Users can also adjust the colour theme of our program using IntelliJ settings.

Principle 5: Tolerance for Error

* We catch any improper input by the user so the program does not crash after receiving such and provide error message telling the user that their input is incorrect and ask them to try again
* All of the possible actions the user can perform along with what they do are presented to the user at each stage of the program and the user only needs to enter a single word or single number input to select them. This minimizes the error that may arise from the need for the user to type in a long specific line of what they want to do.

Principle 6: Low Physical Effort

* All of the input required from user are simple one word input or one number input (except for posting information which the users are free to customize any length of input they would like to make) so that the user does not need to enter a long and complex line of what they want to do

Principle 7: Size and Space for Approach and Use

* Since our program runs in the command line of IntelliJ, the users can use any presentation method that is compatible with presenting IntelliJ. Aside from a presentation method, the only other hardware needed is a keyboard or any other alternative that would allow the user to input letters/numbers into the program (for example use voice recognition software that will convert voice command to input). Thus, the users can customize any presentation method and input method they like to fulfill their needs as long as the methods are compatible with IntelliJ.

We would market our program towards learners who can comprehend simple sentences and have the adequate hardwares needed for presenting and inputting the command line. Since our program mainly aids users in learning new skills and sharing their learning resources to others, people who use our program would most likely be those who would like to learn something or share something they learned. However, it is not limited to any demographic since the new skill the users can learn can be anything from coding, math, to makeup. As long as the users are willing to learn, they are welcomed to use our program as an aid.

Our program is less likely to be used by those who have difficulty comprehending simple sentences. Since our interface is a command line with text interface in simple sentences, those who have difficulty comprehending simple sentences would not be able to understand the interface and thus unable to use the program. In addition, our program is less likely to be used by those who cannot see/read and rely on softwares to convert text to other forms such as voice. Although by using other softwares, the text portion can be converted to other forms for the users to interpret better, these converted forms may not be as clear and understandable as the one visualized in text, especially for the part of the program that visualizes the tree. Thus, although these people can still use our program by using other aiding softwares in combination, the information may not be presented as clearly to them and therefore they would be less likely to use our program.