CSE205: OA: Object-Oriented Program & Data

Date of Reflection:

Oct 11, 2025 6:00 AM

Area of Focus:

This reflection is for my honors CS project. I created this project with anticipation to make it somehow “grab” the data from canvas, and auto schedule the tasks. This way I could sort the assignments, essays, and tasks correctly.

I had a stretch goal to implement an android interface but I was unable to meet this “stretch goal”

The project focuses on the following area:

- Data import
    - Should be able to take tasks from canvas and schedule them
    - Ended up resulting in ICS file importation
- Object-Oriented Design
    - Abstract Classes
    - Interfaces
    - Encapsulation
    - Inheritance
    - Polymorphism
- Data Structure
    - Linked List
    - Queues
    - Stacks
    - Binary Search Tree
- Algorithms
    - Merge Sort
    - Binary Search
- Text Interface
    - Console based output menu

How Did you Implement the Solution? And thoughts?

First off, this project was a lot harder than I thought it would be. I went into this thinking that I could leverage my Python background to implement a data scraper that captures data from the Canvas page, sorts it through various data structures, and prints the data back to the user. Little did I know, it would be quite a bit harder (or dare I say different). So, going through StackOverflow, Reddit, and even looking at some of the documentation from Apple and Google helped me come up with the idea of using ICS, which I would have never guessed could accomplish the same goal. Through this project, I believe my perspective on how to engineer programs like this has shifted significantly. I learned how to use ICS and really learned to utilize the file tools that Java has built into the util package import.

Now, let's get into the actual development solution that I came up with. During the implementation process, I encountered several challenges related to data structure management, generics, and file persistence, which were necessary to ensure the application's stability and core functionality. The first major challenge I had was with data compatibility. I found that the easiest way to implement my solution was to import the ICS file rather than create some revolutionary data scraper (if that's even possible in java). During the development of this part, I created three structures: SimpleLinkedList, MergeSort, and various utility methods. MergeSort requires an array of specific (T[]) that implements the Comparable function. Simply converting the list to a raw object[] would fail, resulting in runtime errors. So my solution was to implement a generic method named public T[] toArray(T[] a) within the linked list class that I created earlier. This design ultimately allowed the calling code in Main to pass an array of the required (new Item[allItems.size()]), which the list then populates and returns. This solution maintains type safety and provides us with the correctly needed array for efficient sorting. This technical bridge was the most crucial to solve to deliver the required sorted agenda view to the user.

The second challenge I encountered, which required a somewhat rigorous implementation, was ensuring data integrity within the FlatFileStore.java class. Dealing with time-based properties was especially difficult. Many schedulable objects, such as tasks and assignments, have optional due dates or end times that can be represented as null or Instant objects. That said, here lies another issue. A failure to handle this null state or Instant during file output could lead to application crashes or inscription corruption. So, my solution was to use an Epoch Second method for persistence and implement a sentinel value to signify a missing date

or time. I found that, specifically, within methods like Task.toFlatFileString(), using ternary operators checks the Instant field and, in the end, saves the numeric value to something like Long.MAX_VALUE instead of 0 for missing/null fields. Consequently, another challenge that I had was when data is loaded; the parsing logic in FlatFileStore checks for sentinel values, and if Long.MAX_VALUE, for example, is encountered, it will correctly assign null rather than erroring out.

Lastly, I had to work around the parsing system as a whole. Finding the proper date-time format wasn't necessarily challenging, but was more so annoying and tedious.

What Did you Struggle With?

What I struggled with the most was the Save/Load functions. Unlike the previous data structures that I had implemented, which directly managed objects purely in memory, save/load required live Java objects within the structure of flat text files. I opted to use a .txt for saving and loading the data because, for me, it was the easiest. The core part of the struggle was with serialization of the .txt file. I found that reliably converting complex objects, such as Tasks, into a single string for file storage and then deserializing that exact string without error was exceptionally challenging. I'm still not 100% sure if the methods/functions I implemented are 100% foolproof, but for now, I consider it a win because they work!

Saving uses a pipe as a delimiter and serializes all of the custom object data in a somewhat predictable order. Loading, on the other hand, was a lot harder for me to accomplish. I found that it required ordered parsing, and every line had to be split and individually checked.

I also found that I struggled with handling Instant objects used for times and dates. The data field could sometimes be null, which led to crashes. My solution, as I stated above, was that it saved and would inscribe as Long.MAX_VALUE, and then when loaded, it was interpreted as no date.

What Went Well?

This part will be brief, as I am somewhat proud of what I have created, and will otherwise say that it all turned out really well.

Personally, I think the sorting mechanism that I implemented was really good. By making the Item hierarchically implement Comparable and then correctly using Merge sort, I was able to accomplish exactly what I wanted to for the sorted agenda part.

The only thing I'm really sad about is that I wasn't able to make an interface, but I will continue developing this app to hopefully make something good out of it!

What Could Have Been Improved?

I will make this part a sort of roadmap or rapid-fire bullet points segment to give some ideas of what I think can be improved upon.

- Data Structure Refinements
    - Overall I would like to implement or work towards implementing the following:
        - Tail pointer in SimpleLinkedList
        - I would like to decouple Node
        - The biggest thing is I would like to actually utilize the queue. Currently the text menu in main only prints items due soon rather than actually enqueuing them in the queue we built. I originally wanted to put the queue to use, but I think I will work towards actually using it in the next version of this that I make.
        - Also I want to try using the stack. In retrospect, I should have just used the queue but… I guess I kind of failed in demonstrating the use of both the queue and the stack as I didn't use either outside of creating them. Maybe we could turn the stack into an undo feature?
- Abstraction and Generics
    - I would like to reduce casting in FlatFileStore. Specifically the Load function currently uses a switch statement followed by a constructor call which I think can be redone using a *Factory Pattern*?? I just read about it so im not 1000% sure, but I think we could use it to reduce the burned on the items constructor!
- UX
    - I would like to implement some sort of input validation
    - A better method of handling File I/O Errors
    - And Error-Proof data parsing…

Time Chart

- Sorry for the gibberish comments. I am writing this after the fact because I was mostly keeping track on my paper notes that I have on my desk!

Action Item	Target Date	Notes	Time Spent (in Hours)
Research	Sep 3, 2025	When I received the Contract files, I immediately began researching solutions. I was initially thinking in Python about how I could do it, and I thought Java would have something similar, so this took a while to wrap my head around. I knew I wanted to implement something like this though as its something I want to actually develop and use myself!	3.5 Hours on and off throughout Sep 3rd and Sep 4th.
Researching 2.0 & Central Planning	Sep 17, 2025	I found out about ICS, and started planning mostly how I could implement it…	2 Hours
Core Structure Implementation	Sep 27, 2025	Created Node.java, SimpleLinkedList.java and added addFirst(), removeFirst(), isEmpty(), and toArray().	3 Hours. I was really struggling here…
Additional Structure Development	Sep 28, 2025	Implemented SimpleStack.java, SimpleQueue.java, and created some of the simpler logic. Also tried to tackle isEmpty().	1 Hours
OOP Model Definitions and Planning	Sep 30, 2025	Defined the Schedulable.java interface and wrote the abstract Item.java class.	1 ½ Hours roughly…
Domain Object and Utilities	Oct 1, 2025	Fixed and wrote a lot of new stuff in Task.java, Assignment.java, Exam.java	2 Hour
Persistence Logic	Oct 2, 2025	Created the FlatFileStore.java. It gave me PTSD	4 Hours… I really couldn't figure it out and kept having tons and tons of errors.

Action Item	Target Date	Notes	Time Spent (in Hours)
Persistence Debugging (;( )	Oct 3, 2025	Bugfixed more errors that popped up…	2 Hours
ICS Integration and Main Application	Oct 4, 2025	Tried to wrap it up here by implementing the ICSParser.java and working with the correct DateTimeFormatter	4 Hours
Bugfixing	Oct 8, 2025	Fixed a lot of the bugs that were happening; fixed errors with importing; fixed errors with loading; fixed errors with DateTimeFormatter	2 Hours
Review, Documentation & Demo	Oct 10, 2025	Created this document, filled out a lot of the info, checked the code again, and prepared a demo that will be attached below!	2.5 Hours
Total:	Oct 11, 2025	Notes: I think this could have gone a lot quicker if I had paid more attention to the text book readings tbh.	~27.5 Hours

Instructions:

Step 1: Open the file in an IDE such as IntelliJ or CS Code.

- Make sure you open it as a project folder!!
- Make sure the Data file is with the project folder (Should be inside).

Step 2: Run the program!

- Menu options:

0) Exit
1) Add Task
2) List Agenda (Sorted)
3) Complete Task by Index
4) Build “Today” Queue (Next 24 hours)
5) Import from ICS file
6) Save
7) Load

- Add task will add a specific task that the user wishes. It will prompt you for title, notes, due date, and priority.
- List Agenda: Shows all items sorted by due date or start time
- Complete Task: Marks a selected task as done
- Build Today Queue: Displays only items due in the next 24 hours
- Import from ICS: Data Import from a canvas or cal .ics file
- Save/Load will save to the local data file

Step 3: Saving and Loading Data

- All data is stored in a text file name studyflow.txt
- To preserve progress, choose “Save” before exiting.
- To restore progress, make sure to choose “Load”

Fileformat:

- TASK|title|notes|dueEpoch|priority|completed
- ASSIGNMENT|title|notes|course|dueEpoch|points
- EXAM|title|notes|course|startEpoch|endEpoch

Importing Events? Read below!

- To import events, download your Canvas cal .ics file
- Document the path. For the easiest method, please put it in downloads or in the project folder.
- Choose option 5 and specify where the file is located.