

FINAL PROJECT HANDbook

Introduction to OOP

Survival Guide For DJJJJ



March 23, 2025

DJJJ

NSCC Ecampus Programing Year 1

**Names:** Dawson Brown, Judah Csanyi, Joshua Leslie, Jeremy Paruch

**Student Numbers:** W0468898, W0509673,W0228010, W0222971

**Due Date:** April 11, 2025

**Course:** PROG 1400 - Introduction to Object-Oriented Programming

**Instructor:** Nadia Gouda

Table of Contents

[Introduction to OOP 0](#_Toc195264043)

[Project Overview 0](#_Toc195264044)

[The Presentation 0](#_Toc195264045)

[What do we need for this project 1](#_Toc195264046)

[Operations for Each User Type 1](#_Toc195264047)

[File Handling for Data Storage 1](#_Toc195264048)

[Exception Handling & Validation 1](#_Toc195264049)

[Use of Java Collections & OOP Principles 1](#_Toc195264050)

[(Bonus) Graphical User Interface (GUI) Implementation 1](#_Toc195264051)

[How the heck do we pass this in? 2](#_Toc195264052)

[The Code 2](#_Toc195264053)

[The UML Diagram and project documentation 2](#_Toc195264054)

[The Proof of Concept 2](#_Toc195264055)

[The Group Policy (Ethos) 3](#_Toc195264056)

[Commitment to Participation 3](#_Toc195264057)

[Respect and Communication 3](#_Toc195264058)

[Accountability 3](#_Toc195264059)

[Quality 3](#_Toc195264060)

[Support 3](#_Toc195264061)

[The Group Policy Continued (Ethos) 4](#_Toc195264062)

[Conflict Resolution 4](#_Toc195264063)

[Transparency 4](#_Toc195264064)

[Work-Life Balance 4](#_Toc195264065)

[Decision-Making Process 4](#_Toc195264066)

[We will conduct group votes before making decisions. The majority will decide; in the case of a tie, the designated tiebreaker will make the final decision on the course of action. 4](#_Toc195264067)

[Who is taking care of what? 5](#_Toc195264068)

[Dawson 5](#_Toc195264069)

[Jeremy 5](#_Toc195264070)

[Josh 5](#_Toc195264071)

[Judah 5](#_Toc195264072)

[Speculation of Project Development 6](#_Toc195264073)

[Week 1: 6](#_Toc195264074)

[Week 2: 6](#_Toc195264075)

[Week 3: 6](#_Toc195264076)

[Week 4 April 13-16) 6](#_Toc195264077)

[Project Documentation Notes 7](#_Toc195264078)

[March 27, 2025 7](#_Toc195264079)

[March 28, 2025 7](#_Toc195264080)

[March 31, 2025 7](#_Toc195264081)

[March 31, 2025(Continued) 8](#_Toc195264082)

[April 01,2025 10](#_Toc195264083)

[April 02, 2025 11](#_Toc195264084)

[April 03, 2025 11](#_Toc195264085)

[April 03, 2025( Continued) 12](#_Toc195264086)

[April 04, 2025 12](#_Toc195264087)

[April 07,2025 13](#_Toc195264088)

[April 08, 2025 13](#_Toc195264089)

[April 10, 2025 13](#_Toc195264090)

[April 11 13](#_Toc195264091)

[Challenges Throughout the Project 17](#_Toc195264092)

Table of Figures

[Figure 1: 3 Arrays 8](#_Toc195264093)

[Figure 2: Try and catch when creating a file 8](#_Toc195264094)

[Figure 3:Feeding the file to the array lists 8](#_Toc195264095)

[Figure 4:Reading the file and object is not an employee object 9](#_Toc195264096)

[Figure 5: corrected menu class 11](#_Toc195264097)

[Figure 6: Option 4 fixed 12](#_Toc195264098)

[Figure 7: Example of discovered Bug 13](#_Toc195264099)

[Figure 8: Fix try and catch 14](#_Toc195264100)

[Figure 9: Fixing Salary if you try to enter a string 15](#_Toc195264101)

[Figure 10LCode showing how it works 16](#_Toc195264102)

# Introduction to OOP

## Project Overview

This is a basic management system to handle students, instructors (faculty) and employees, we are developing a structured application to store and manage RECORDS and using OOP to do it

**We MUST submit a “Proof of Concept 2 DAYS before presentation, aka a working prototype.**

## **The Presentation**

Will be going over the demonstration of the program itself, showing the UML design and technical discussion

# What do we need for this project

Basic User Management

* Store and manage students, instructors, and employees.
* Attributes: name, ID, email, and role (student, instructor, or employee).

## Operations for Each User Type

* Students: Register students and display student list.
* Instructors (Faculty): Store faculty details and display them.
* Employees: Store non-teaching staff records and display them.

## File Handling for Data Storage

* Save student, instructor, and employee data to a file.
* Load saved data when the program starts.

## Exception Handling & Validation

* Prevent duplicate records.
* Validate inputs (e.g., IDs must be unique).

## Use of Java Collections & OOP Principles

* Implement inheritance, encapsulation, and polymorphism.
* Use ArrayLists or HashMaps to manage records

## (Bonus) Graphical User Interface (GUI) Implementation

* Use Java Swing or JavaFX for a basic UI to add and display records.

# How the heck do we pass this in?

## The Code

A fully functional Java program with a menu-driven system for managing students, instructors, and employees.

✔ Organized into appropriate packages.

✔ Uses OOP principles (inheritance, polymorphism, encapsulation).

✔ Implements file handling for storing data.

✔ Includes exception handling for user input errors.

✔ (Bonus) GUI implementation using Java Swing or JavaFX.

We send the source code files

We also send screenshots the program is working

## The UML Diagram and project documentation

A UML class diagram showing system structure.

A short document explaining:

* The system working
* How each OOP concept was used
* Challenges we had to deal with and how did we solve it

We need the UML diagram in either png jpg or pdf

Documentation is either a docx or pdf

## The Proof of Concept

To reiterate this has to be passed in TWO DAYS before the final presentation, the prototype has to demonstrate:

* Adding and displaying student records
* Adding and displaying instructor and employee records
* Basic file handling (saving and loading data)

# The Group Policy (Ethos)

## Commitment to Participation

We will make an effort to contribute equally to our projects by attending meetings and fulfilling our responsibilities in tasks.

## Respect and Communication

We aim to promote open and respectful communication so that every team member feels comfortable sharing their ideas, concerns, and feedback.

## Accountability

Each of us is accountable for completing our assigned tasks in a timely manner. If an unexpected situation arises, it’s essential to inform the team early and collaborate on finding a solution.

## Quality

Our goal is to deliver high-quality work instead of hurrying, as rushing could lead to errors.

## Support

We will actively support each other by sharing our knowledge and offering help when someone is facing difficulties.

# The Group Policy Continued (Ethos)

## Conflict Resolution

In the event of disagreements, team members should tackle conflicts with a focus on solutions and aim for a compromise; if this fails, faculty intervention may be necessary.

## Transparency

We will keep each other informed and updated on our progress regarding the projects.

## Work-Life Balance

We will strive to avoid overworking ourselves and acknowledge the significance of taking breaks.

## Decision-Making Process

# We will conduct group votes before making decisions. The majority will decide; in the case of a tie, the designated tiebreaker will make the final decision on the course of action.

# Who is taking care of what?

## Dawson

* Programming
* Lead Speaking

## Jeremy

* Documentation, power point building/presenter
* Main Speaker, for presentation
* Requesting Bropheus programming
* Bug Catcher

## Josh

* Programming,
* Building Powerpoint,
* Diagram (?)
* documentation(?), t
* Troubleshooting, prettying it, try and catch

## Judah

* Wants to program, codesplaining

# Speculation of Project Development

## Week 1:

Day 1: Kickoff, assign tasks, create shared task board/tracker

Day 2-3 Work on OOP

Day 4-5: continue on OOP for two hours, switch to starting client side programming for two hours

Day 6-7: OOP build continues, assign light tasks for Linux project and C project

## Week 2:

Day 8-10: Finish Project 1 Proto Type (April 2nd)

Day 11-14 Full Sprint Client Side Development (Target prototype April 05)

Parllel Coding/set up for C Project

## Week 3:

Days 15-16 Push C project to April 09 deadline

Days 17-18 Work on Linux Project (april 12 deadline)

Day 19-20 (Review ALL PROTOTYPES, Start presentation prep)

## Week 4 April 13-16)

Refine All Projects

Fix Bugs

Finish Documentation

Rehearse Presentations

|  |  |  |
| --- | --- | --- |
| Project | Prototype Deadlines | Buffer/Polish |
| OOP | April 02 | April 07-13 |
| Client Side | April 05 | April 07-13 |
| C Project | April 09 | April 10-13 |
| Linux | April 12 | April 13-16 |

# Project Documentation Notes

## March 27, 2025

* Person class = your basic   
  Student class =subclasses major, gpa,
* Instrutor=subclasses  
  Employees=ubclasses Employee has additional variables start date enddate and salary and job.
* Variables have been established
* To get time format we need java.time.LocaDate
* For ID’s Student gets S Instructor gets I employee gets E

## March 28, 2025

* UML Diagram started by Josh.
* Variable Format issue discovered, constructors are currently set to public
* Class Person should changed to User.
* Needs cardinality and which relationship arrows to use.

## March 31, 2025

* We have 3 array lists one for students, one for instructors, one for employees

## March 31, 2025(Continued)

Figure : 3 Arrays

A computer screen shot of text

AI-generated content may be incorrect.

* We write to a file using object.input surrounded in a try and catch. The try is catching any errors happen when writing to the file. The file is saved into a .dat file.

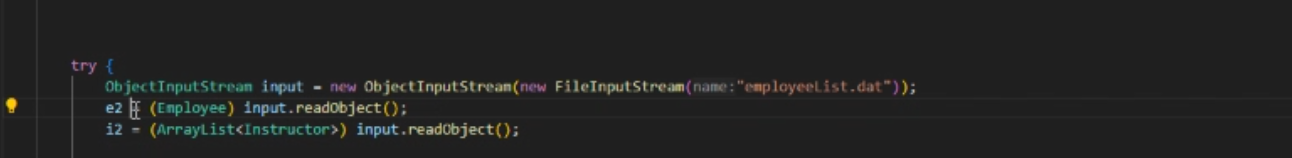
Figure : Try and catch when creating a file

A computer screen with colorful text

AI-generated content may be incorrect.

When we start the program we the file is fed into the empty array lists when the program starts up

Figure :Feeding the file to the array lists



This way whenever the program starts a new if there is a file available, the program can read the file and load the data into each established array list

Figure :Reading the file and object is not an employee object

A screenshot of a computer

AI-generated content may be incorrect.

# April 01,2025

Judah believes a few menu options are required, group agrees

Main menu:

1) Manage Students

1.1) Add Student

1.2) Display Details of all students

1.3) Search for student by Name

1.4) Search for student by

1.4.1) Change Name

1.4.1) Change Age

1.4.1) Change Major

1.4.1) Change Address

1.4.1) Change GPA

1.4.1) return to menu

1.5) Delete Student

1.6) Return to menu

2) Manage Employees

3) Manage Instructors

4) Exit

# April 02, 2025

Menu class was misbehaving, Judah and Josh worked together to fix the issue by having it be an object instead of being called directly, now its menu.mainMenu();

Figure : corrected menu class

A black background with white text

AI-generated content may be incorrect.

Jeremy finished the bulk of the Powerpoint Presentation, Storyboard also done

# April 03, 2025

1. The program operates by allowing three types of users: students, instructors, and employees. It begins with the main function, which opens a file and creates an instance of the `menuManagement` class. The main function then calls “menuManagement.mainMenu()”, which initializes a “menuProcessor” object.
2. Through “menuManagement.mainMenu()”, the main menu is displayed, and user input is collected. Depending on the input, the program directs the user to the appropriate submenu, such as the instructor, student, or employee menus. The “menuManagement.studentMenu” utilizes the “menuProcessor” to effectively manage the inputs for the student-related actions.
3. The “menuProcessor” is needed for the programs functionality. In order to run the program, users must compile the Java files using the command `javac \*.java` followed by executing the main class with `java Main.java`, The F5 Function does not work.

# April 03, 2025( Continued)

Found an issue with the Java program. The problem comes from the creation of new entities such as students, employees, or instructors. When closing the program and then reopening it, the identifiers for these new entries reset, everythings is gone. For example, if a student is created with an ID like S1000, and then the program is closed, upon reopening and creating another student, the new student will again receive the same ID S1000. This leads to duplicates in the IDs.

# April 04, 2025

Yesterday's Issue was resolved, now it doesn’t start over at S1000 when you open students from the database, but uses the last ID from the students in the array list

Option 4 still doesn’t work, employees, students and instructors all share the same ID counter. When a student is created with an ID of s1000, and then you create an employee, it will be E1001

\*Edit\* Option 4 works, the solution was copying the id search for each array list and modifying it

Figure : Option 4 fixed

A computer screen shot of text

AI-generated content may be incorrect.

# April 07,2025

UML Diagram completed by Josh

# April 08, 2025

Team Meeting Check in. Discussions are in the notes

UML Diagram updated, missing menuprocessing class

# April 10, 2025

Jeremy Documented some Bug report documents and finalized, bugs located and were dealt with applying try( catch(exception e){ please use proper input type )

Figure : Example of discovered Bug

A computer screen shot of a computer code

AI-generated content may be incorrect.

# April 11

Judah fixed various bugs from the bug testing on April 10th. Examples provided

Figure : Fix try and catch

A screen shot of a computer program

AI-generated content may be incorrect.

Figure : Fixing Salary if you try to enter a string

A screen shot of a computer program

AI-generated content may be incorrect.

Figure : Code showing how it works

A computer screen shot of a program

AI-generated content may be incorrect.

# Challenges Throughout the Project

* Managing saving to the file was hard and took a bit of time to figure out
* The inputs for the Instructor, Student, and Employee were all using the id counter from the User class, and so they would be counting off each other rather than their own separate counter
* Making sure the id counters started at the last object created from the class was hard because I had to scan the array list, take the last object in the array, and manually update the id counter when the program is first run
* Finding Bugs, a few mistakes here and there with Booleans