## Name – Josaiah Nassi and Gael Sustaita

## Date – April 22, 2025

CS 3331 – Advanced Object-Oriented Programming – Spring 2025

## Instructor – Bhanukiran Gurijala

## Assignment – Project Part 1 Lab Report

This work was done as a team and completely on our own. We did not share, reproduce, or alter any part of this assignment for any purpose. We did not share code, upload this assignment online in any form, or view/received/modified code written from anyone else. All deliverables were produced entirely on my/our own. This assignment is part of an academic course at The University of Texas at El Paso and a grade will be assigned for the work we produced.

# **Program Explanation**

This assignment involved developing a space debris tracking system for Low Earth Orbit (LEO). Our program simulates interactions with orbital data for multiple different user types including scientists, space agency reps, policymakers, and administrators. Each user role has distinct functionalities, such as tracking debris, assessing orbital risk, generating reports, and managing user access.

To tackle the problem, we broke it down into classes one for each user role. We then created a class hierarchy using abstract base classes and inheritance. We used encapsulation to protect internal data and modularized logic into separate classes such as OrbitAssessment, ImpactAnalysis, and UserManager. Techniques such as file I/O handling, role-based menu navigation, and persistent logging were used to simulate a realistic and modular console application.

# **What did I learn?**

This assignment helped us improve on our understanding of advanced object-oriented programming principles such as inheritance, abstraction, and encapsulation. We also gained some more experience with Java file handling, user interaction design, and implementing logging mechanisms.

We realized that our solution could be improved with more robust error handling and perhaps a graphical interface for ease of use. Another possible approach would be to integrate a database for managing users and debris data more effectively.

This project took us approximately 25–30 hours to complete over multiple sessions. The most time-consuming part was ensuring each user role’s functionality worked independently before integration.

# **Solution Design**

The program was designed as a modular Java application with a clear separation of concerns. The MissionControl class serves as the central controller, while individual functionality was divided across specialized classes like TrackingSystem, OrbitAssessment, ImpactAnalysis, and UserManager.

We used the following data structures:

* ArrayList for storing all space objects
* HashMap for orbit type-based density reporting
* BufferedReader and BufferedWriter for file operations

Assumptions made:

* The system was created heavily based on rso\_metrics.csv (the file provided), so the system assumes any other input files are formatted the same
* Data directories (data/, logs/) exist or are created dynamically

# **Testing**

We used a combination of testingfocusing on user interface behavior and testing examining internal logic such as orbit evaluation conditions.

To test the system:

* We created custom rso\_metrics.csv data with edge cases such as very old debris or missing orbit types.
* We verified menu functionality for each user type
* Validated output files (updated\_orbit\_status.csv, exited\_debris\_report.txt)
* Reviewed the log file to confirm actions were recorded accurately

Testing can be improved by writing automated unit tests using JUnit and mocking file inputs.

Test Cases Used:

* Debris with and without valid longitude/orbit
* Creating and deleting users as Admin
* Viewing risk levels before and after assessment as Policymaker
* Generating density from different orbit types as Agency

# **Test results**

The testing showed that all user menus were accessible and functional as intended. Selecting "Scientist" allowed for both object tracking by category and accurate orbit assessment. We were also able to create and delete a user with the name joey. The assessment produced two valid output files: a CSV summarizing risk levels and orbit status, and a TXT listing exited debris.

A black screen with white text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# **Code Review**

* **Implementation**: We ensured the program performs the correct functions per user role. Each action produces expected outputs, and logic for orbit risk classification is derived directly from the problem statement.
* **Logic**: The program handles expected and edge-case inputs gracefully. Invalid menu entries are caught, and missing files produce warnings without crashing.
* **Readability/Style**: We used descriptive class/method names, consistent indentation, and added Javadoc to each public class and method. Code blocks were modular and logically grouped.
* **Maintainability**: Each user role's logic is encapsulated in its own class, making the system extendable. If new roles or actions are added, only their respective class would need updates.
* **Performance**: For this dataset size, the linear search and file handling logic performed efficiently. Most collections used are optimal for their purpose (ArrayList for lists, HashMap for counts).