Name

Caitlin Gregory
Daniela Gutierrez

Date

April 20, 2025

CS 3331 – Advanced Object-Oriented Programming – Spring 2025

Instructor

Dr. Bhanukiran Gurijala

Assignment

Project Part 1

1.

What Did We Do?

We began by individually drafting UML diagrams to outline the architecture of the system. This included creating a **Use Case Diagram** to visualize the different user roles and their interactions with the system, followed by a **Class Diagram** that structured the classes and relationships required to fulfill the specified functionality.
Based on the Class Diagram, we implemented several core classes, starting with the **file-reading pipeline**:

- `FileLoader` reads the CSV file line by line.
- `ArrayBuilder` converts each line into a 2D array.
- `DictionaryBuilder` parses the array and instantiates the appropriate subclass of `SpaceObject` based on the object type (e.g., `Satellite`, `RocketBody`, `Debris`,

`Payload`, or `UnknownObject`). All instances are stored in a `LinkedHashMap` keyed by their record ID.

We then created a `RunSimulation` class to contain the `main` method. This class initializes the user interface by calling the `DisplayMenu` class, which presents a dynamic menu based on the selected user type.

The `DisplayMenu` class connects to `MissionControl`, which is responsible for coordinating the tracking system. Upon instantiation, `MissionControl` loads and stores the parsed object data so that users can interact with it. Currently, the system fully supports the Scientist user type. The Scientist menu provides two major features:

- **Track Objects in Space**, which allows filtering by type (e.g., Rocket Body, Debris, Payload, Unknown).
- **Assess Orbit Status**, which determines if debris is still in orbit and evaluates orbital drift to assign a risk level.

How Did We Tackle the Problem?

To manage complexity, we took a modular approach. The entire system was broken down into manageable components:

- **User Interface Handling** (`DisplayMenu`)
- **Data Input and Parsing** (`FileLoader`, `ArrayBuilder`, `DictionaryBuilder`)
- **Object Modeling** (`SpaceObject` and its subclasses)
- **Core Functionality** (`MissionControl`)
- **Logging and File Management** (`Logger`)

Each component was designed to perform a specific task and interact with others through well-defined interfaces. This made it easier to develop, test, and debug each part independently.

What Techniques Did We Use?

- **Object-Oriented Programming**: We used inheritance to model different types of space objects and user roles.
- **Encapsulation and Abstraction**: Data-related logic is encapsulated within specialized classes like `MissionControl` and `Logger`, reducing coupling.
- **File I/O**: We read and parsed CSV data efficiently, using defensive programming to handle missing or malformed fields.
- **Data Structures**: We used a `LinkedHashMap` for ordered storage and constant-time access to space object records.
- **Modular Design**: Functions and responsibilities were clearly separated, improving clarity and scalability.

Did We Break the Problem into Smaller Problems?

Yes. Given the broad scope of the project, we intentionally decomposed it into smaller problems:

- Parsing raw data from files
- Mapping objects based on type
- Building a dynamic and role-based menu system
- Handling user interaction and logging
- Implementing Scientist-specific features first, with the infrastructure in place for future extension

By following this divide-and-conquer strategy, we ensured steady progress and made debugging more manageable throughout development.

2.

What Did We Learn?

This project deepened our understanding of **object-oriented programming (OOP)** and highlighted the importance of abstraction in managing complex systems. By abstracting space data into higher-level classes such as `SpaceObject` and its subclasses (`Debris`, `Satellite`, `Payload`, etc.), we were able to structure our solution in a scalable, modular, and maintainable way.

We also recognized the value of using the **right data structures**. Specifically, storing our objects in a `LinkedHashMap`allowed for fast lookups while preserving the original input order—making iteration consistent and efficient. This approach contributed to better system performance and improved readability throughout the codebase.

Furthermore, we gained hands-on experience with **modular design principles**, **file input/output (I/O)**, and **persistent logging**, while also learning how to structure user interaction in a real-world context through a role-based menu system.

How Can the Solution Be Improved?

While our implementation successfully supports the Scientist role and meets core requirements, there are areas that can be refined:

- **Code Reusability**: Menu logic—especially repeated `Scanner` objects and `Logger` calls—can be abstracted into shared utility methods to reduce redundancy and simplify updates.
- **Error Handling**: More robust input validation and exception handling would improve system stability and user experience.

Alternative Solutions

An alternative design could involve replacing the CSV-based input system with a **relational database**. Using a database would improve data integrity, support scalability, and allow for more complex queries and filters across sessions.

In addition, implementing **design patterns** like Factory (for creating space objects) or Strategy (for user role behaviors) could further modularize the system and make it more adaptable to future changes.

How Long Did It Take?

This lab assignment was completed over the full duration between the release and the submission deadline. Time was distributed across several key phases:

- Drafting and revising UML diagrams
- Implementing file parsing and data modeling
- Designing and coding the menu system
- Conducting testing and debugging
- Logging activity, documenting functionality, and writing the lab report

3.

What Did We Do in This Program?

In this program, we built a modular, menu-driven Java application to track and analyze space debris and satellite data in Low Earth Orbit (LEO). The system supports various user roles—including Scientists, Space Agency Representatives, Policymakers, and Administrators—with role-specific functionality. We focused on implementing full support for the Scientist role, which includes tracking objects in space and assessing their orbital status.

We began by designing UML diagrams to visualize our system and plan its architecture. From there, we implemented file reading and parsing logic, object instantiation based on CSV data, and a main menu system that directs users through different interactive options.

What Was Our Approach?

Our approach was to **divide the problem into smaller, manageable components**, each with a clear responsibility:

- **Data Loading**: Load and parse CSV input.

- **Data Modeling**: Create space object classes and assign object types using inheritance.
- **Interaction Handling**: Build a role-based menu system using `DisplayMenu`.
- **Processing and Logic**: Implement functionality such as object filtering and orbit status analysis in `MissionControl`.

This modular design allowed us to develop and test each piece independently while maintaining a consistent structure across the system.

What Data Structures Did We Use, and Why?

We used a **`LinkedHashMap<String, SpaceObject>`** to store parsed space objects, indexed by their unique `recordId`. This data structure was selected for two main reasons:

1. **Constant-Time Access (O(1))**: The dictionary allows us to retrieve any object by its ID in constant time, which is efficient for lookup-based operations.
2. **Preserved Insertion Order**: The `LinkedHashMap` maintains the order in which elements were added, which aligns with how users might expect to see the data presented (matching the original CSV order).

This combination of speed and ordering made it an ideal choice for our needs.

What Assumptions Did We Make?

To simplify development, we made the following assumptions:

- The input CSV file is properly formatted and contains all the expected columns.
- Object types (e.g., “Debris”, “Rocket Body”, “Payload”) are accurately labeled in the dataset.
- The user provides valid numerical input for menu selections. While we included basic error messages, full exception handling is a planned future improvement.
- The system only needs to support one user session at a time and logs persist across sessions via file appending.

These assumptions helped streamline our implementation and allowed us to focus on core functionality.

4. Testing

We tested the program using both **black-box** and **white-box** testing strategies.

- **Black-box testing** was used to verify the menu navigation and user input interactions by running the program in the console and selecting various options.
- **White-box testing** was applied when checking file parsing and internal data flow, particularly by creating a `TestRunSimulation.java` file to inspect whether the CSV data was correctly read and stored in the appropriate object classes.

We tested the core functionalities—such as displaying tracked objects and assessing orbit status—by manually reviewing outputs for each object type. We also intentionally gave invalid inputs to ensure the menu displayed appropriate error messages.

Although the program passed all functional tests, testing practices can be improved by automating test cases using a framework like **JUnit** and mocking user input/output interactions. Future improvements could also include testing edge cases like empty files, malformed CSV entries, or repeated IDs.

Test Cases Used:

- Load the CSV file and print parsed objects
- Run through each user role menu path
- Input invalid menu choices (e.g., letters, out-of-range numbers)
- Track each type of object: Rocket Body, Debris, Payload, Unknown
- Assess orbit status and validate log creation
- Exit the program and confirm output files and logs are saved

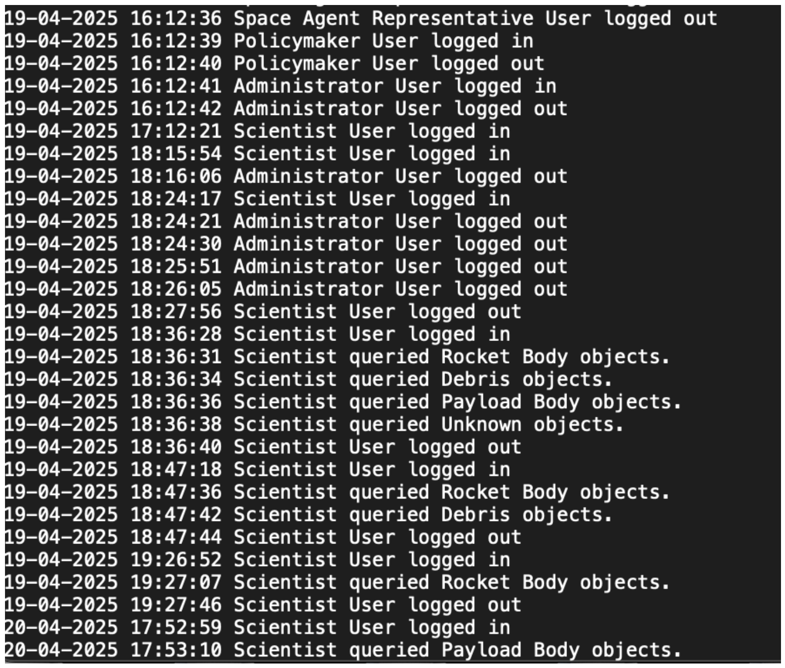
We also attempted to break the program intentionally by providing invalid file paths and malformed input. These failures helped us adjust file reading logic and improve exception handling.

5. Test results

The results from our tests confirmed that:

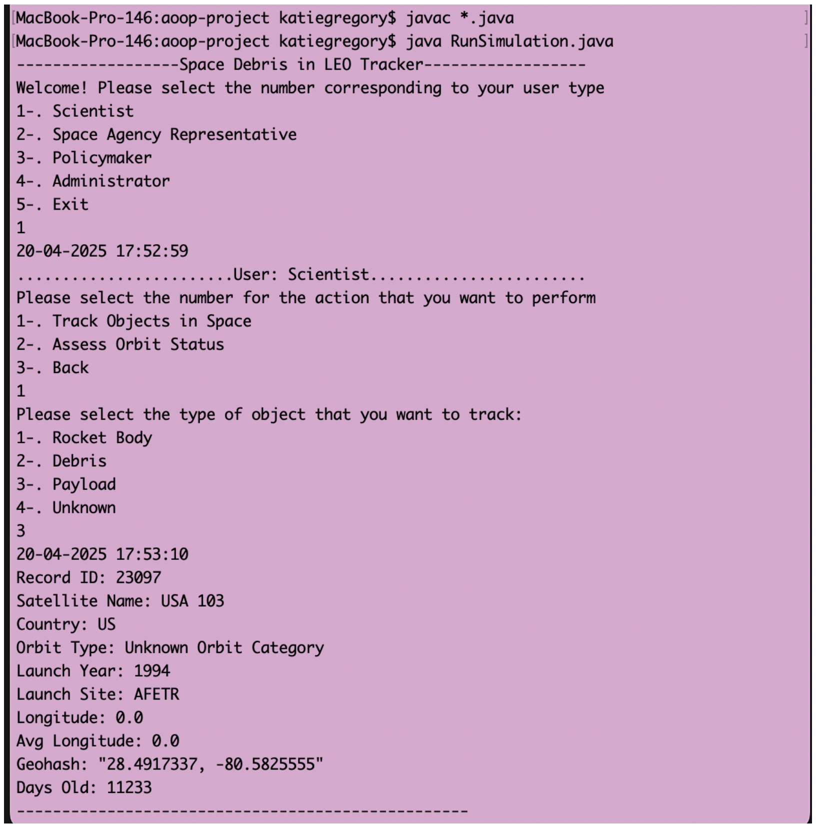
- The CSV file was read successfully and parsed into appropriate `SpaceObject` subclasses.
- User menus were displayed correctly based on user role input.
- Objects were accurately listed when selecting tracking options.
- Risk levels were calculated correctly based on orbital drift.
- The `logger.txt` file recorded actions persistently across runs.

Sample `logger.txt` output:



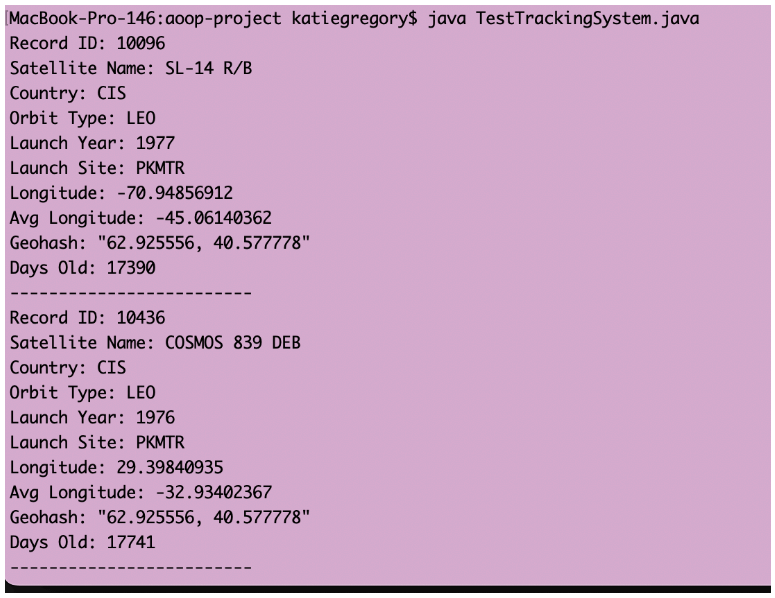
```
19-04-2025 16:12:36 Space Agent Representative User logged out
19-04-2025 16:12:39 Policymaker User logged in
19-04-2025 16:12:40 Policymaker User logged out
19-04-2025 16:12:41 Administrator User logged in
19-04-2025 16:12:42 Administrator User logged out
19-04-2025 17:12:21 Scientist User logged in
19-04-2025 18:15:54 Scientist User logged in
19-04-2025 18:16:06 Administrator User logged out
19-04-2025 18:24:17 Scientist User logged in
19-04-2025 18:24:21 Administrator User logged out
19-04-2025 18:24:30 Administrator User logged out
19-04-2025 18:25:51 Administrator User logged out
19-04-2025 18:26:05 Administrator User logged out
19-04-2025 18:27:56 Scientist User logged out
19-04-2025 18:36:28 Scientist User logged in
19-04-2025 18:36:31 Scientist queried Rocket Body objects.
19-04-2025 18:36:34 Scientist queried Debris objects.
19-04-2025 18:36:36 Scientist queried Payload Body objects.
19-04-2025 18:36:38 Scientist queried Unknown objects.
19-04-2025 18:36:40 Scientist User logged out
19-04-2025 18:47:18 Scientist User logged in
19-04-2025 18:47:36 Scientist queried Rocket Body objects.
19-04-2025 18:47:42 Scientist queried Debris objects.
19-04-2025 18:47:44 Scientist User logged out
19-04-2025 19:26:52 Scientist User logged in
19-04-2025 19:27:07 Scientist queried Rocket Body objects.
19-04-2025 19:27:46 Scientist User logged out
20-04-2025 17:52:59 Scientist User logged in
20-04-2025 17:53:10 Scientist queried Payload Body objects.
```

Sample Console Output (Tracking Payloads):



```
MacBook-Pro-146:aoop-project katiegregory$ javac *.java
MacBook-Pro-146:aoop-project katiegregory$ java RunSimulation.java
------------------Space Debris in LEO Tracker------------------
Welcome! Please select the number corresponding to your user type
1-. Scientist
2-. Space Agency Representative
3-. Policymaker
4-. Administrator
5-. Exit
1
20-04-2025 17:52:59
.......................User: Scientist.......................
Please select the number for the action that you want to perform
1-. Track Objects in Space
2-. Assess Orbit Status
3-. Back
1
Please select the type of object that you want to track:
1-. Rocket Body
2-. Debris
3-. Payload
4-. Unknown
3
20-04-2025 17:53:10
Record ID: 23097
Satellite Name: USA 103
Country: US
Orbit Type: Unknown Orbit Category
Launch Year: 1994
Launch Site: AFETR
Longitude: 0.0
Avg Longitude: 0.0
Geohash: "28.4917337, -80.5825555"
Days Old: 11233
--------------------------------------------------
```

Sample Console Output (TestTrackingSystem - CSV File Parsing):



```
MacBook-Pro-146:aoop-project katiegregory$ java TestTrackingSystem.java
Record ID: 10096
Satellite Name: SL-14 R/B
Country: CIS
Orbit Type: LEO
Launch Year: 1977
Launch Site: PKMTR
Longitude: -70.94856912
Avg Longitude: -45.06140362
Geohash: "62.925556, 40.577778"
Days Old: 17390
-------------------------
Record ID: 10436
Satellite Name: COSMOS 839 DEB
Country: CIS
Orbit Type: LEO
Launch Year: 1976
Launch Site: PKMTR
Longitude: 29.39840935
Avg Longitude: -32.93402367
Geohash: "62.925556, 40.577778"
Days Old: 17741
-------------------------
```

6. Code Review

To ensure the quality and completeness of our implementation, we conducted an individual code review using the provided checklist. Each member of the team reviewed the following areas:

- **Code Structure and Style**: Verified consistent formatting, naming conventions, and modularization.
- **OOP Principles**: Checked that inheritance, abstraction, and encapsulation were properly applied.
- **Documentation**: Reviewed comments and verified the presence of Javadoc headers in all public classes and methods.
- **Error Handling**: Ensured try-catch blocks were used for file operations and that input validation was present where necessary.
- **Functionality**: Manually verified that each method performed its intended task.

We also looked out for redundant code and opportunities to refactor, especially within the menu logic. The review process helped us identify and address areas where reusability could be improved and where input validation could be enhanced.