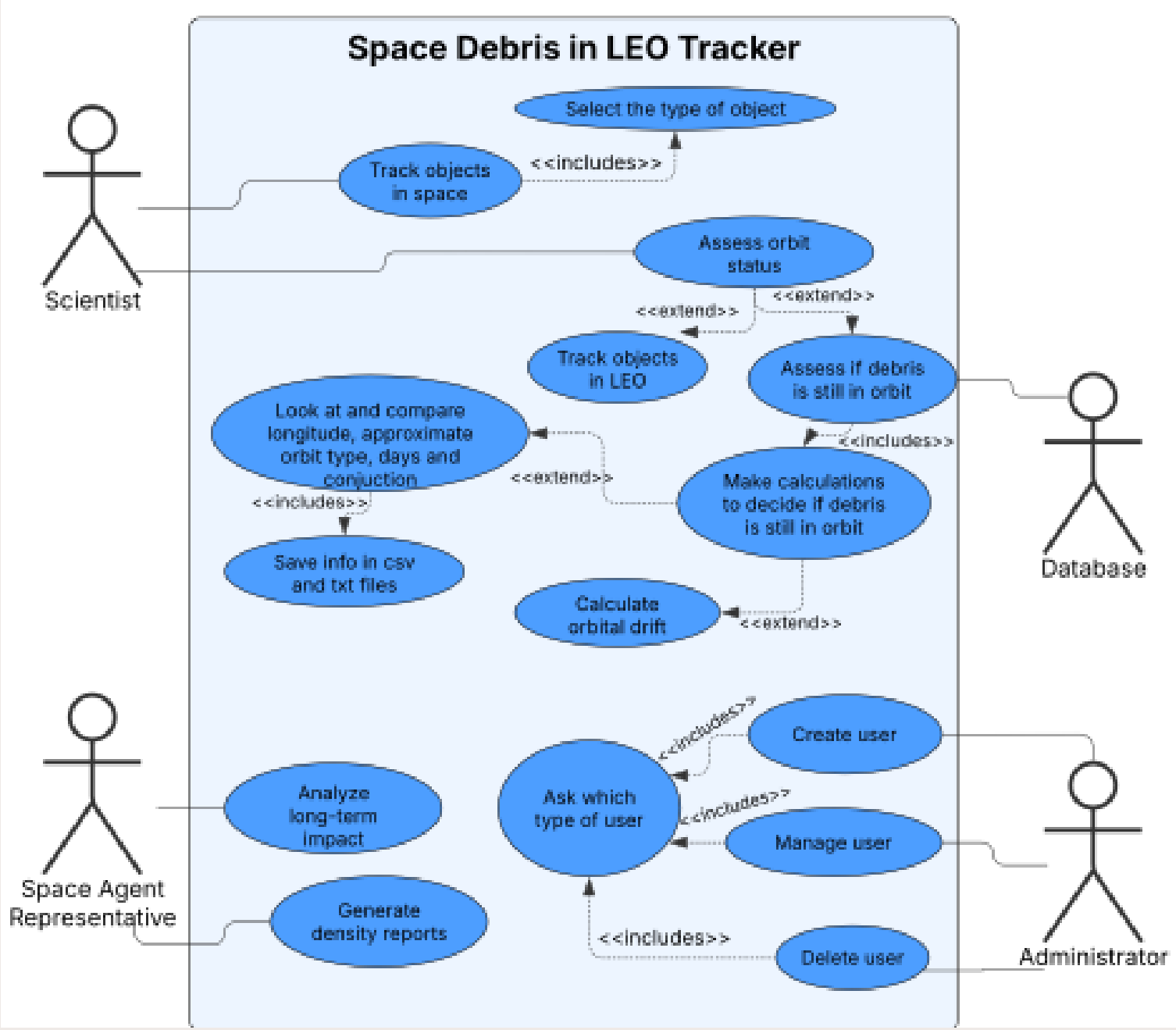
# CS 3331

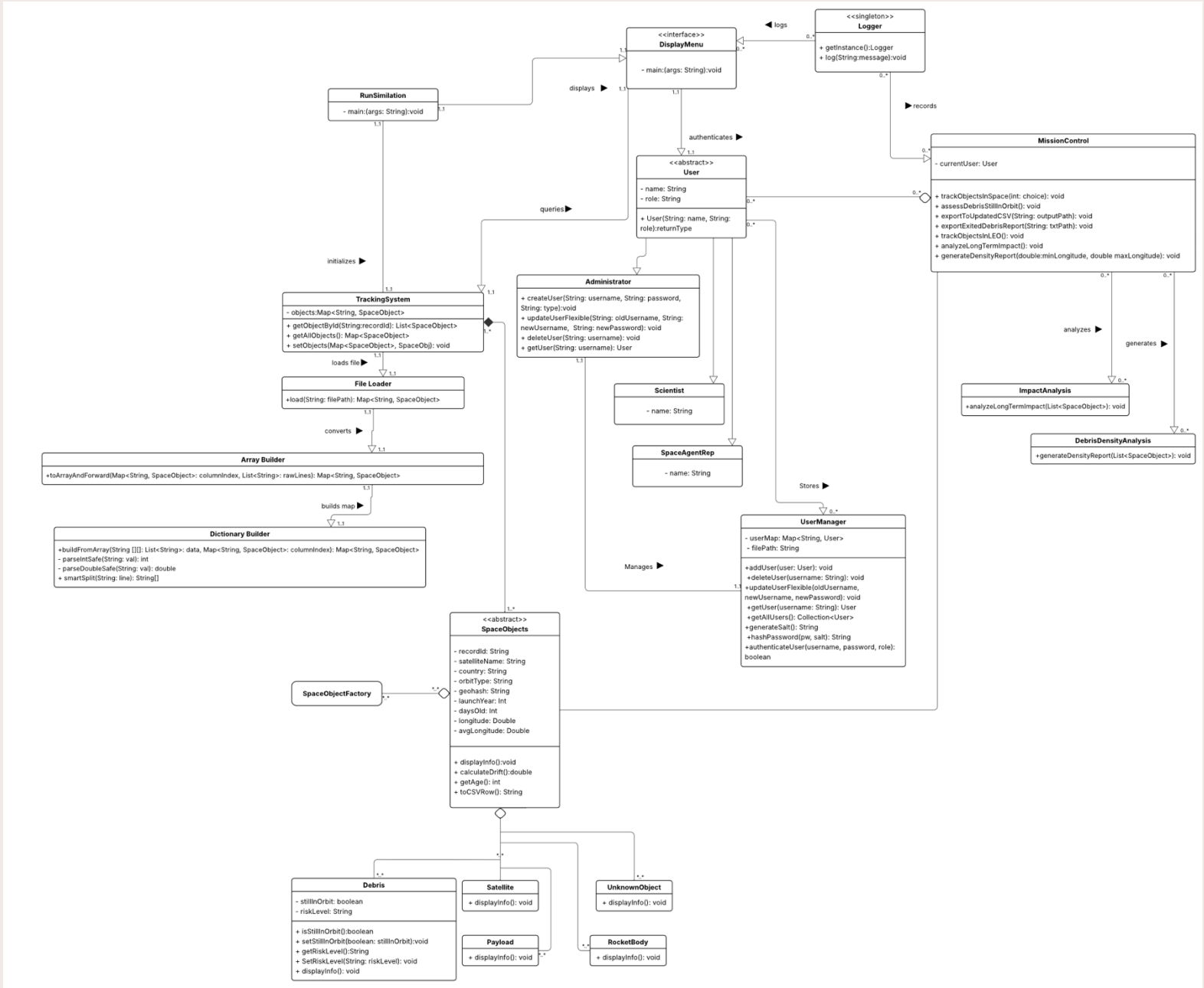
# Space Debris in LEO

Team Members:
Caitlin Gregory
Daniela Gutierrez

# Diagrams

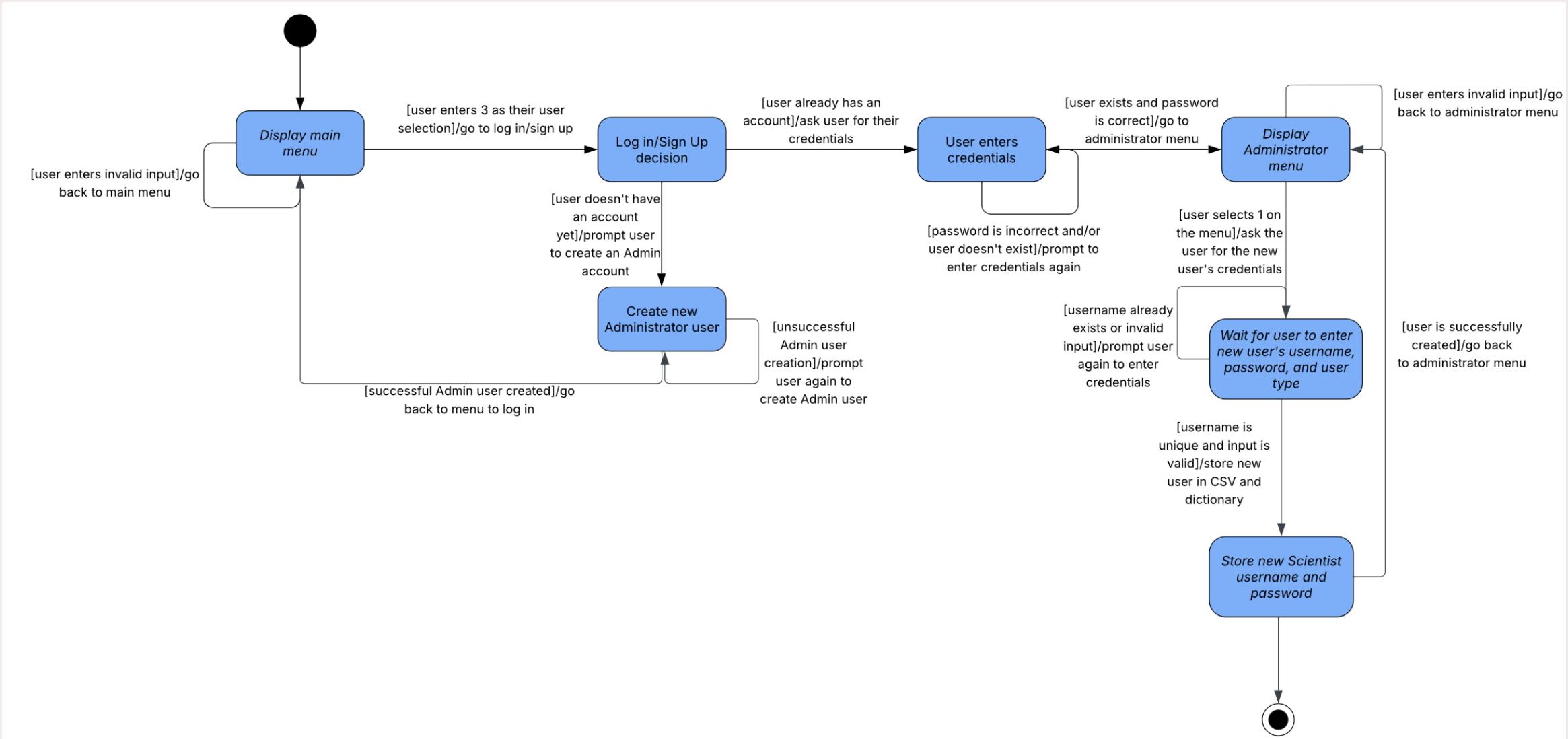
# UML Use Case Diagram



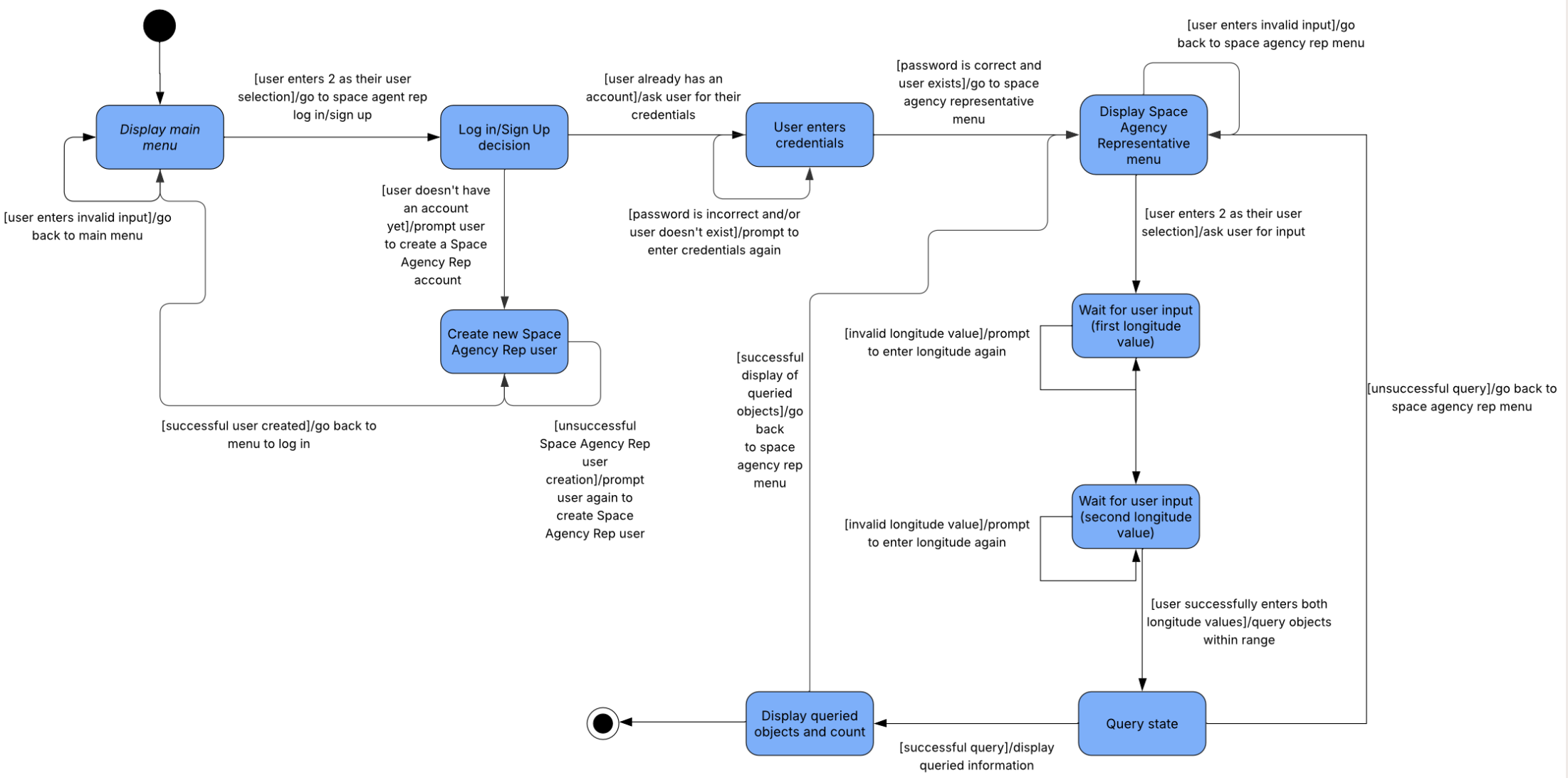



# UML Class Diagram

# UML State Diagram
# Add New Scientist User



# UML State Diagram
## Generate Density Reports



# How did we use Object-Oriented Programming?

Our project uses object-oriented programming principles extensively to promote modularity, reusability, and scalability:

- **Abstraction:** High-level classes like MissionControl, DisplayMenu, and UserManager encapsulate complex functionality behind simple interfaces.
- **Encapsulation:** Data and behavior are tightly coupled in classes such as Debris, RocketBody, and User, ensuring safe access through methods like setRiskLevel() or authenticateUser().
- **Polymorphism:** Methods like displayMenu() and trackObjectsInSpace() operate on base types (User, SpaceObject) but invoke subclass-specific behavior seamlessly (e.g., Debris vs Payload).
- **Inheritance:** User is a base class extended by Scientist, Administrator, and SpaceAgentRep, allowing role-specific behavior while sharing common logic.

# How did we use Design Patterns?

- We implemented the **Singleton Design Pattern** in our **Logger** class.
- In a system with multiple menus and user interactions, consistent and centralized logging is essential. We wanted to:
  - Avoid creating multiple logger instances.
  - Ensure all logs write to the same place (logger.txt file).
  - Promote memory efficiency and prevent redundancy.
- The Logger class has a **private static instance** and a **private constructor**, preventing external instantiation.
- It exposes a getInstance() method to provide global access to the one and only logger.
- The components (like DisplayMenu, MissionControl, and user classes) call Logger.getInstance().log(...) to log activity in a consistent way, passing in a String with the desired message to log.

```java
public class Logger{
    // 1. Private static instance
    // Only instance of Logger
    private static Logger instance;

    // 2. Private constructor
    // prevents other classes from creating a new logger
    private Logger(){}

    // 3. Public static method to get the single instance
        // Controls access to the only instance.
        // The first time it's called, it creates the instance.
        // On every later call, it just returns the already-created one.
    public static Logger getInstance(){
        if (instance == null){
            instance = new Logger();
        }
        return instance;
    }
```

```java
    // Logging print method
    public void log(String message){
        LocalDateTime myDateObj = LocalDateTime.now();
        DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern(pattern:"dd-MM-yyyy HH:mm:ss");
        String formattedDate = myDateObj.format(myFormatObj);

        String appendText = "\n" + formattedDate + " " + message;

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName:"logger.txt", append:true))){
            writer.write(appendText);
        } catch (IOException ioe){
            System.out.println(x:"Couldn't write to file");
        }
    }
```

# JUnit

To produce test cases for the JUnit, we thought about all (or at least most) of the possible input scenarios for the specific code section that we were testing, and we used assertions to determine whether the code was returning the expected results.

```java
class MissionControlTest {

    private MissionControl missionControl;
    private TrackingSystem trackingSystem;
    private final ByteArrayOutputStream outContent = new ByteArrayOutputStream();
    private final PrintStream originalOut = System.out;
    private UserManager userManager;

    /**
     * Prepares the test environment before each test.
     * Initializes test user manager, tracking system, and test debris data.
     */
    @BeforeEach
    void setUp() {
        System.setOut(new PrintStream(outContent));

        trackingSystem = new TrackingSystem();
        trackingSystem.setObjects(new HashMap<>());
        missionControl = new MissionControl(trackingSystem);

        userManager = new UserManager(filePath:"test_users.csv");
        userManager.getAllUsers().clear();

        String salt = userManager.generateSalt();
        String hash = userManager.hashPassword(password:"password123", salt);
        User user = new Scientist(username:"testUser", hash, salt);
        userManager.addUser(user);

        trackingSystem.getAllObjects().put(key:"qualifying", new Debris(recordId:"9999", satelliteName:"TestSat", country:"US", orbitTyp…"LEO",
                longitude:45.0, avgLongitude:45.0, geohash:"geo", hrrCategory:"HRR", …false, false, false, 300, 5));
    }

    /**
     * Cleans up after each test by resetting output and removing test files.
     */
    @AfterEach
    void tearDown() {
        System.setOut(originalOut);
        outContent.reset();
        new File(pathname:"test_users.csv").delete();
    }
```

# JUnit

```java
/**
 * Tests generation of a debris density report within a specified longitude range.
 * Verifies that only objects in range are written to the report file.
 * Uses a backup file to avoid overwriting actual production data.
 */
@Test
void testGenerateDensityReport() throws IOException {
    File realFile = new File(pathname:"density_report.csv");
    File backupFile = new File(pathname:"density_report_backup.csv");

    if (realFile.exists()) {
        if (backupFile.exists()) backupFile.delete();
        realFile.renameTo(backupFile);
    }

    try {
        if (realFile.exists()) realFile.delete();

        trackingSystem.getAllObjects().put(key:"A", new Debris(recordId:"A", satelliteName:"Debris-A", country:"US", orbitTyp…"LEO", 2020, "
            longitude:20.0, avgLongitude:0.0, geohash:"hash1", hrrCategory:"HRR", …false, false, false, 100, 1));
        trackingSystem.getAllObjects().put(key:"B", new Debris(recordId:"B", satelliteName:"Debris-B", country:"US", orbitTyp…"LEO", 2020, "
            longitude:150.0, avgLongitude:0.0, geohash:"hash2", hrrCategory:"HRR", …false, false, false, 100, 1));
        trackingSystem.getAllObjects().put(key:"C", new Debris(recordId:"C", satelliteName:"Debris-C", country:"US", orbitTyp…"LEO", 2020, "
            longitude:500.0, avgLongitude:0.0, geohash:"hash3", hrrCategory:"HRR", …false, false, false, 100, 1)); //out of range

        missionControl.generateDensityReport(minLongitude:10, maxLongitude:200);

        assertTrue(realFile.exists(), "density_report.csv should be created");

        List<String> lines = java.nio.file.Files.readAllLines(realFile.toPath());
        assertTrue(lines.stream().anyMatch(line -> line.contains(s:"Debris-A")), "Should contain Debris-A");
        assertTrue(lines.stream().anyMatch(line -> line.contains(s:"Debris-B")), "Should contain Debris-B");
        assertFalse(lines.stream().anyMatch(line -> line.contains(s:"Debris-C")), "Should NOT contain Debris-C");

    } finally {
        if (realFile.exists()) realFile.delete();
        if (backupFile.exists()) backupFile.renameTo(realFile);
    }
}
```

```java
/**
 * Tests user login with various valid and invalid combinations of username, password, and role.
 */
@Test
void testLogin() {
    boolean success = userManager.authenticateUser(username:"testUser", password:"password123", role:"Scientist");
    assertTrue(success, "Login should succeed with correct credentials and role");

    success = userManager.authenticateUser(username:"testUser", password:"wrongPass", role:"Scientist");
    assertFalse(success, "Login should fail with incorrect password");

    success = userManager.authenticateUser(username:"testUser", password:"password123", role:"Administrator");
    assertFalse(success, "Login should fail with incorrect role");

    success = userManager.authenticateUser(username:"ghost", password:"password123", role:"Scientist");
    assertFalse(success, "Login should fail for non-existent user");
}
```

# Javadoc

The Javadocs provide us with useful documentation about the functionality, classes, variables, and overall functionality of our code; they are generated after writing comments in a specific format.

## All Classes and Interfaces

All Classes and Interfaces | Interfaces | Classes | Exception Classes

| Class | Description |
|---|---|
| Administrator | Class that represents the Administrator user. |
| ArrayBuilder | Converts raw lines into a 2D array and passes it to the dictionary builder. |
| Debris | The Debris class represents a space object classified as debris. |
| DebrisDensityAnalysis | Class that generates the density analysis report |
| DensityReportException | Custom exception for errors encountered while writing the density report. |
| DictionaryBuilder | Builds a dictionary of SpaceObject instances from a 2D array. |
| DisplayMenu | This class handles all menu interactions for all users |
| FileLoader | Responsible for loading the file and initiating the parsing process. |
| ImpactAnalysis | Implement long term impact analysis functionality from space agency rep |
| Logger | This class is responsible for logging all queries, user interactions and updates on the system. |
| Menu | Interface that is implemented in DisplayMenu and RunSimulation, methods are called in DisplayMenu |
| Payload | Class that represents a Payload object. |
| RocketBody | Class that represents a Rocket Body object |
| Satellite | The Satellite class represents a space object classified specifically as a satellite. |
| SpaceObject | The SpaceObject class serves as an abstract base class for objects in space such as Debris and Satellite. |
| SpaceObjectFactory | Factory class for creating specific types of SpaceObject instances based on the object type string from the dataset. |
| TrackingSystem | TrackingSystem is the central class that loads and manages space object data. |
| UnknownObject | This class represents an unknown space object. |
| User | Class that represents a User. |
| UserManager | Manages user accounts including creation, deletion, update, authentication, and persistent storage to a CSV file. |

# Index

A C D F G H I L M O P R S T U
All Classes and Interfaces | All Packages | Serialized Form

## A



## C



## D

# Reflections

- This assignment helped us strengthen our ability using different classes and objects, and how they relate to each other.
- We learned a lot about important Object-Oriented Programming principles such as inheritance, polymorphism, and encapsulation.
- We broke this problem into smaller parts by thinking about the concepts in a high-level manner and then going deeper into the specifics of the implementation.
- We think that this course allowed us to delve deeper into a topic that we first discussed in CS1, and to truly understand how the interactions between objects can compose a complex system.
- One of the issues that we faced was understanding the way the diagrams were supposed to be composed, specially for the state diagram. The file reading was also difficult at times.
- We were able to overcome the challenges presented by asking the instructional team for help and clarification, or by browsing through the Internet to try to find a possible solution.

# References

- https://www.geeksforgeeks.org/sha-256-hash-in-java/ -> for the hashing of the passwords
- https://beginnersbook.com/2014/01/how-to-append-to-a-file-in-java/ for writing into an existing file

# Thank you!