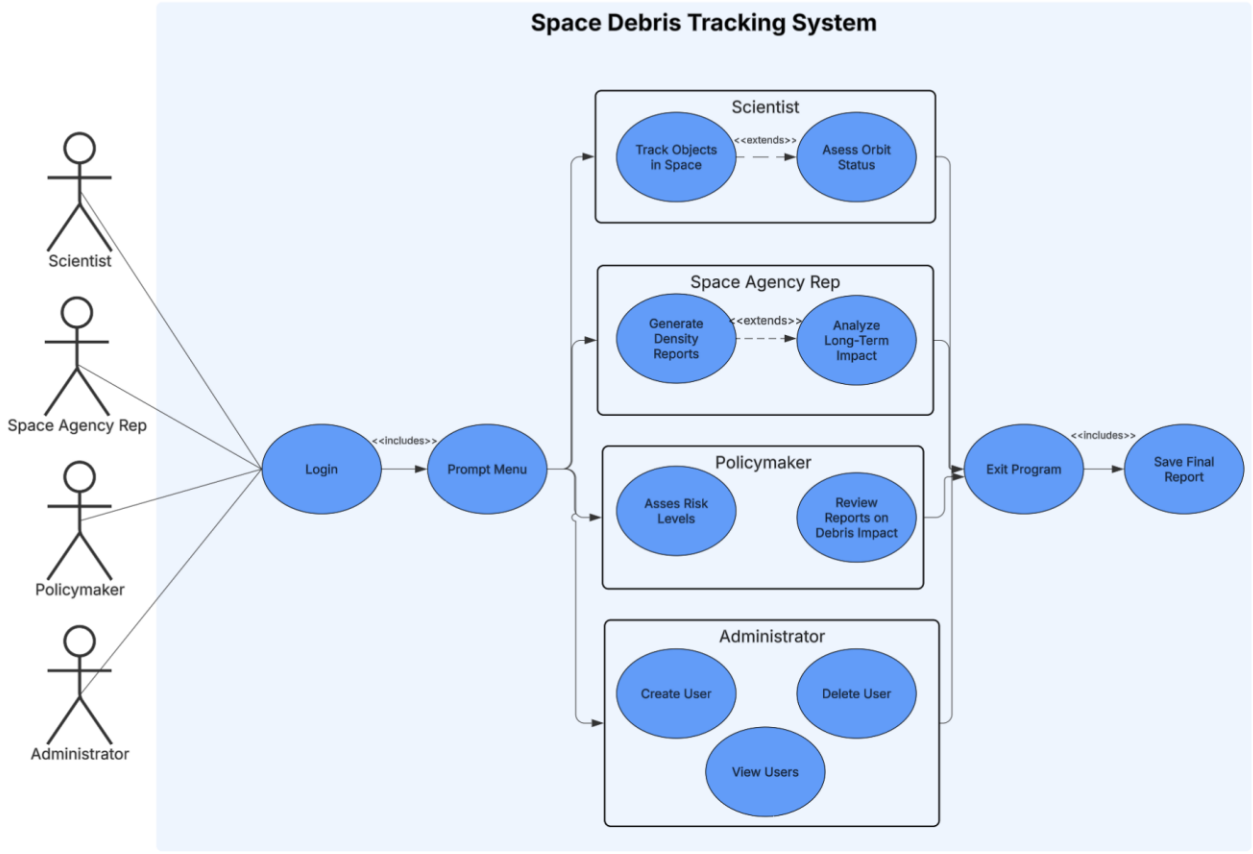
The Java-nauts

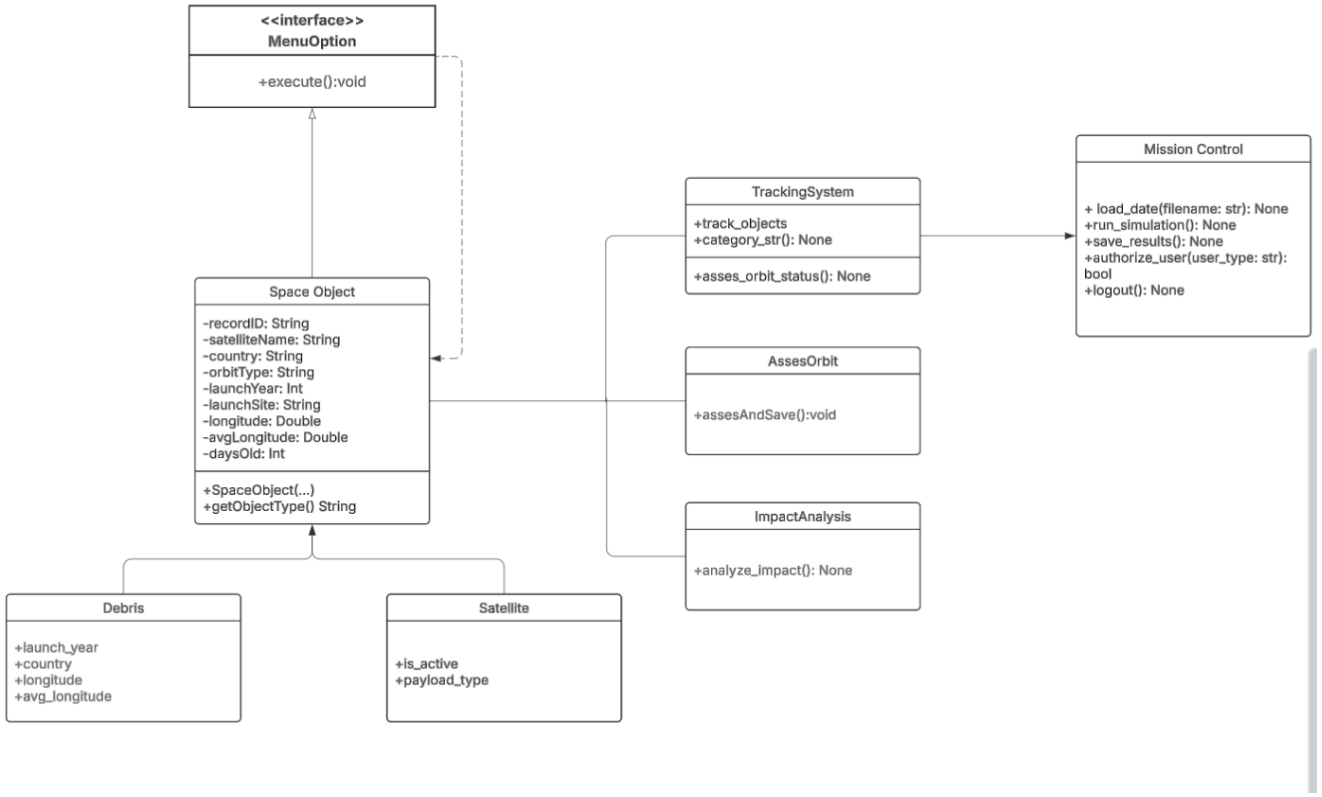
TEAM MEMBERS:

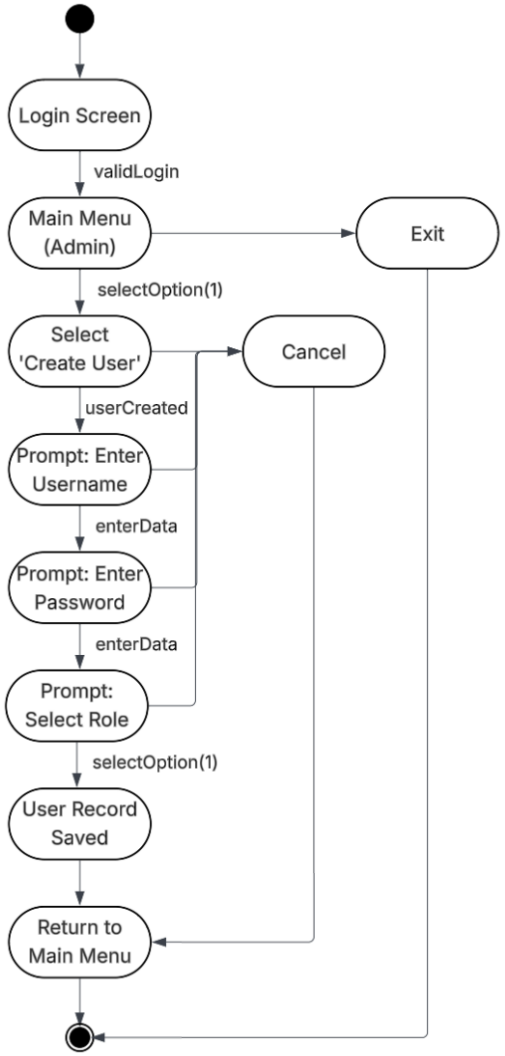
JOSAIAH NASSI AND GAEL SUSTAITA

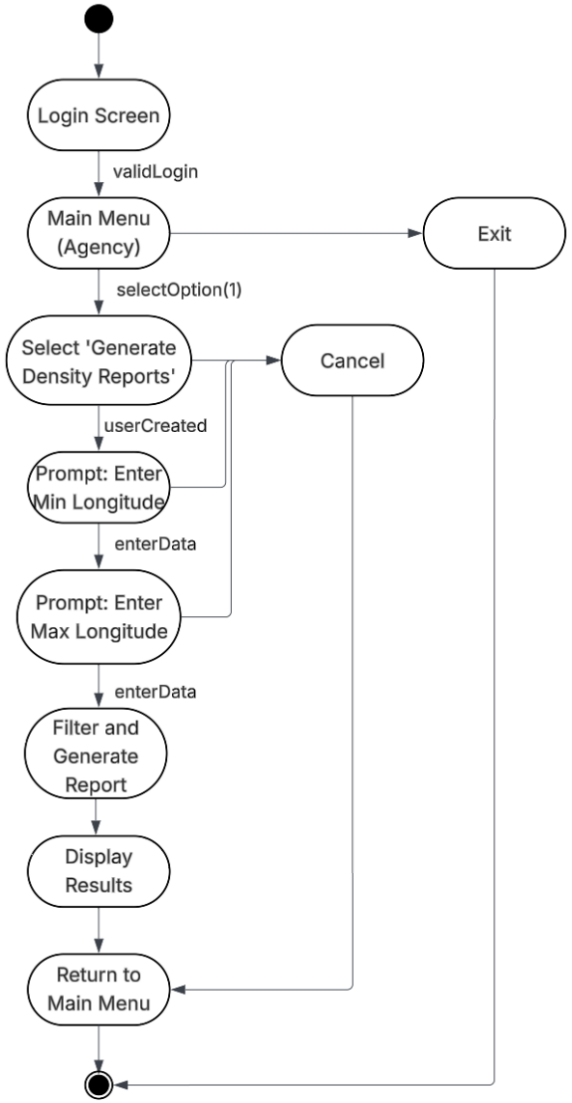
UML Use Case Diagram



UML Class Diagram







UML State Diagram

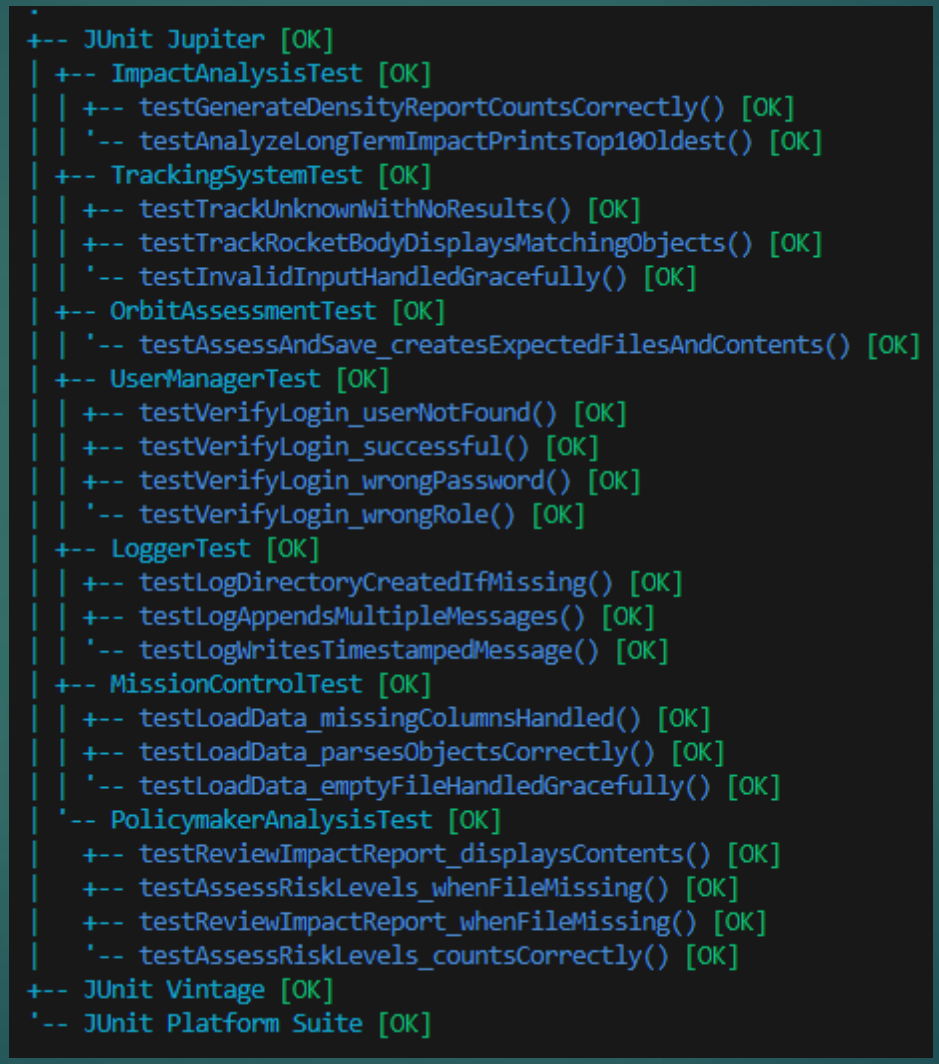
How did we use Object-Oriented Programming?

In our project, we used OOP principles to design a reusable, modular, and maintainable system. We used abstraction in the SpaceObject class, which defined the shared properties and methods for all space objects like debris and satellites. Inheritance was used to create specialized classes like Debris and Satellite, which inherit from SpaceObject. We used encapsulation by controlling access to object attributes and exposing behaviors through clearly defined methods. Polymorphism was used in both method overriding and in the implementation of the Strategy design pattern.

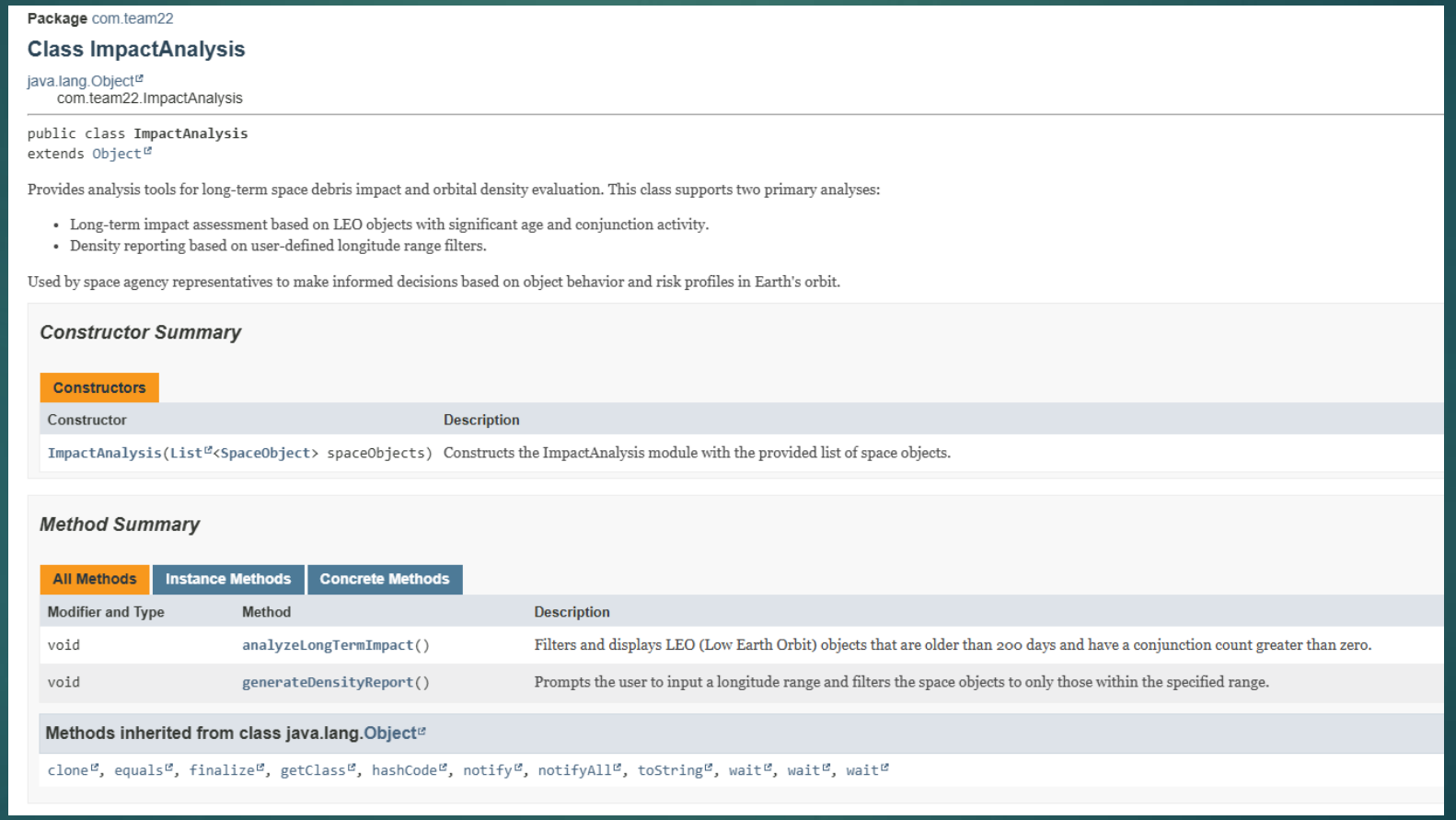
How did we use Design Patterns?

- We incorporated key design patterns to enhance the structure and flexibility of our system. We used the Facade Pattern was applied through the MissionControl class, which acts as the central interface for all user interactions, managing menu navigation and delegating tasks to the appropriate modules. Additionally, we used the Strategy Pattern to handle different reporting behaviors. The ReportStrategy interface allowed us to dynamically switch between long-term impact analysis and orbit assessment, using strategy classes like ImpactReportStrategy and OrbitAssessmentStrategy. These design patterns help us simplify control flow, make behaviors easily replaceable, and promote separation of responsibilities.

JUnit


.
+-- JUnit Jupiter [OK]
| +-- ImpactAnalysisTest [OK]
| | +-- testGenerateDensityReportCountsCorrectly() [OK]
| | '-- testAnalyzeLongTermImpactPrintsTop10Oldest() [OK]
| +-- TrackingSystemTest [OK]
| | +-- testTrackUnknownWithNoResults() [OK]
| | +-- testTrackRocketBodyDisplaysMatchingObjects() [OK]
| | '-- testInvalidInputHandledGracefully() [OK]
| +-- OrbitAssessmentTest [OK]
| | '-- testAssessAndSave_createsExpectedFilesAndContents() [OK]
| +-- UserManagerTest [OK]
| | +-- testVerifyLogin_userNotFound() [OK]
| | +-- testVerifyLogin_successful() [OK]
| | +-- testVerifyLogin_wrongPassword() [OK]
| | '-- testVerifyLogin_wrongRole() [OK]
| +-- LoggerTest [OK]
| | +-- testLogDirectoryCreatedIfMissing() [OK]
| | +-- testLogAppendsMultipleMessages() [OK]
| | '-- testLogWritesTimestampedMessage() [OK]
| +-- MissionControlTest [OK]
| | +-- testLoadData_missingColumnsHandled() [OK]
| | +-- testLoadData_parsesObjectsCorrectly() [OK]
| | '-- testLoadData_emptyFileHandledGracefully() [OK]
| '-- PolicymakerAnalysisTest [OK]
| +-- testReviewImpactReport_displaysContents() [OK]
| +-- testAssessRiskLevels_whenFileMissing() [OK]
| +-- testReviewImpactReport_whenFileMissing() [OK]
| '-- testAssessRiskLevels_countsCorrectly() [OK]
+-- JUnit Vintage [OK]
'-- JUnit Platform Suite [OK]

Javadoc



Package com.team22

Class ImpactAnalysis

java.lang.Object
  com.team22.ImpactAnalysis

```
public class ImpactAnalysis
extends Object
```

Provides analysis tools for long-term space debris impact and orbital density evaluation. This class supports two primary analyses:

- Long-term impact assessment based on LEO objects with significant age and conjunction activity.
- Density reporting based on user-defined longitude range filters.

Used by space agency representatives to make informed decisions based on object behavior and risk profiles in Earth's orbit.

Constructor Summary

Constructors

Constructor	Description
`ImpactAnalysis(List<SpaceObject> spaceObjects)`	Constructs the ImpactAnalysis module with the provided list of space objects.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	`analyzeLongTermImpact()`	Filters and displays LEO (Low Earth Orbit) objects that are older than 200 days and have a conjunction count greater than zero.
void	`generateDensityReport()`	Prompts the user to input a longitude range and filters the space objects to only those within the specified range.

Methods inherited from class java.lang.Object

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Reflections

- This assignment was a valuable learning experience in designing complex software using OOP principles and architecture. Throughout the process, we gained hands-on experience working with inheritance, interfaces, abstraction, and modular design. We learned how to break a large problem into manageable parts by dividing our project into user-specific modules and focusing on one functionality at a time. As CS students, we improved our ability to plan, refactor, and test software in a disciplined way. Some of the biggest challenges we faced included implementing flexible CSV parsing, managing dynamic user input across roles, and understanding design patterns in a real-world context. We overcame these obstacles by researching best practices, testing often, and collaborating closely as a team.

References (If applicable):