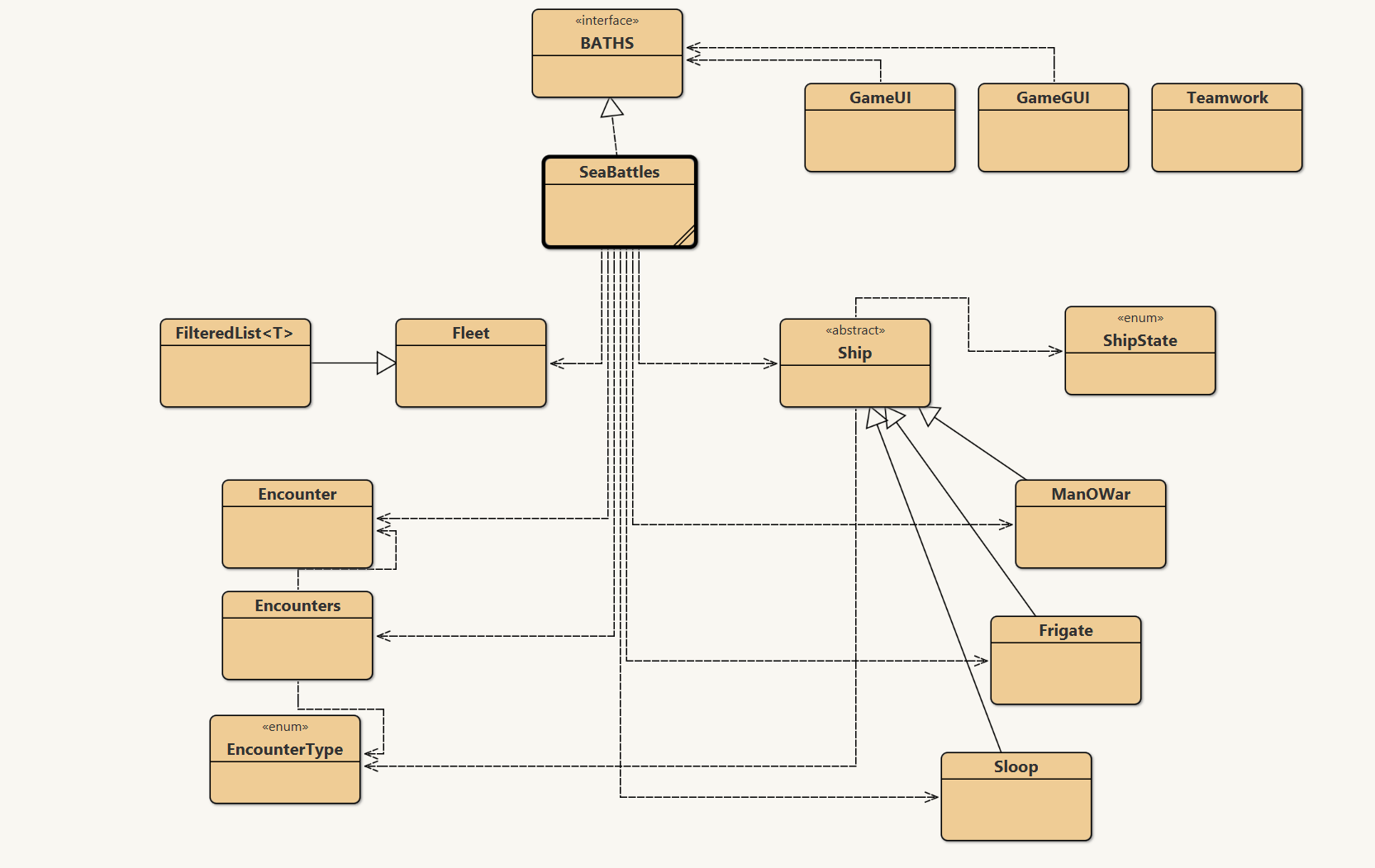
**Technical report for “BATHS” assignment**

**Introduction**

This is the explanation of the key design decisions and implementation plans for the Battles and the High Seas (BATHS) system, developed as part of the “**Principles and Practices of Large-scale programming”** assignment**.** The system needed to be developed strictly adhering to software engineering principles like modularity, low coupling, high cohesion, and test-driven development. Below, we include a UML diagram explaining the system structure followed by an explanation of three key design decisions. Each decision is balanced against other choices, with justifications for the solution chosen based on maintainability, reusability, and functionality.



**Design decision 1:**

**Using Filters on** allShips **Instead of Maintaining Multiple Lists.**

**Reason:**

Originally, separate lists (e.g., for active ships, sunken ships, player ships, etc.) were created in Fleet and maintained in SeaBattles. This was changed to a single allShips list with filtered views to access subsets as needed.

**Alternatives Considered:**

Maintain separate lists for different categories of ships (e.g. activeShips, sunkenShips).

**Pros and Cons:**

**1**.Single allShips with filters(chosen):

**Pros:** Reduces redundancy, consistent state, flexible querying

**Cons:** Slightly more processing time when filtering (minor impact)

**2.** Multiple specific lists:

**Pros:** Faster access to certain types, more direct code in places.

**Cons:** Risk of desynchronization, harder to manage on state updates.

**Why We Chose This:**

This is less redundant and simpler to deal with data. To have all the ships in one list and filtering to search for categories like "sunken" or "active" ships eliminates the synchronizing issue and lets us refer dynamically to any group of ship state or properties without having duplicated data structures.

**Design decision 2:**

**Using a HashMap<EncounterType, Boolean> for Navy Rules and Implementing Encounter Management**

**Reason:**

To maintain the record of the rules of what type of encounters are allowed or conducted, a HashMap<EncounterType, Boolean> was added in the SeaBattles class. A getSize() method was added, and Encounter logic was added in SeaBattles.

**Alternatives Considered:**

1.Use if else statements to select the correct Boolean to check

2.Using external methods to loop and find the correct Boolean

**Pros and Cons:**

**1.** HashMap for rules (chosen):

Pros: Compact, easily extensible, O(1) access.

Cons: Slightly more complex syntax than individual ifelse statments.

**2.** If else statments:

Pros: Simpler implementation.

Cons: Not scalable, less maintanable.

**3.** External encounter manager:

**Pros:** Cleaner modular separation.

**Cons:** Adds overhead, harder to keep logic centralized.

**Why We Chose This:**

To maintain the record of the rules of what type of encounters are allowed or conducted, a HashMap<EncounterType, Boolean> was added in the SeaBattles class. This allowed to easily access the correct Boolean through a single method and use this to check if a Ship can fight the Encounter

**Design decision 3:**

**Improving Performance via HashMap – Eliminating the Need for Loops.**

**Reason:**

We enhanced data retrieval operations by using HashMap data structures instead of relying on for loops iterating over lists. This enabled us to have worst-case constant time (O(1)) access encounters.

**Alternatives Considered:**

1. Continue using ArrayList and iterate through it to find matching entries.

**Pros and Cons:**

**1.** HashMap (chosen).

**Pros:** O(1) lookup time, cleaner logic, scales well

**Cons:** Slightly higher memory usage

**2.** Linear search via loop. (Alternative)

**Pros:** Easier to implement initially.

**Cons:** O(n) performance in worst-case, less maintainable.

**Why We Chose This:**

Substituting lookups with HashMap access made code more efficient and readable in a significant way. It is especially helpful since the number of game objects grows, ensuring the same performance regardless of the number of stored objects.

**Conclusion:**

In all stages of development of BATHS system, we in our team have ensured clean, scalable, and efficient design. We refactored the ship management approach by collapsing all ships into a single list and applying filters, keeping the program considerably simpler but gaining flexibility and robustness. For managing encounter rules, we included a HashMap of EncounterType values as keys and their activation state as the value this allows for better future expansion for the game and simple maintainability and readability in the program. Also, iterative search code replacement with HashMap lookups improved performance and made the code easier. Each one of these design choices reflects our focus on readability, maintainability, and optimization. Not only did they improve existing functionality in our system but also set us up for future potential features. The result is a clean, stable SeaBattles implementation that follows good object-oriented practice and meets the project's technical needs well.