5.1 Code Review

This code review focuses on the query function in my QueryAvailable class, responsible for selecting appropriate drones and the service point they are available at based on medical delivery requirements.

Correctness and Functional Behaviour:

The function correctly implements the required filtering logic for drone selection. It validates the drone capabilities (heating, cooling and capacity), availability against date and time constraints on deliveries, and cost constraints relative to delivery requirements. Null checks were used to reduce the risk of runtime errors. Overall, the function satisfies requirements.

Design and Maintainability:

A main weakness of the implementation if that the function performs many distinct logical tasks within a single function. This includes capability filtering, availability filtering, capacity checking and cost calculation. This violates the Single Responsibility Principal and makes the function difficult to read, maintain and test in isolation. Refactoring the function into smaller helper methods with individual responsibilities would improve clarity and reduce complexity.

Readability and Structure:

The function relies on deeply nested loops and Boolean flag variables to control flow. While functionally this is correct, readability is significantly reduced and makes the logic harder to understand. In several cases, early returns could simplify control flow. Variable naming is generally good and descriptive but occasionally doesn't follow typical Java naming conventions.

Style and Implementation:

Several Style and implementation issues were identified:

- Magic numbers were used and embedded directly into calculations, rather than defined as constants, reducing readability and maintainability.
- Sections of commented or unused debugging code remain and should be removed or replaced with formal logging if required.
- Lack of comments, reducing maintainability and readability.

Performance:

The function contains many heavily nested loops that scale poorly with increasing numbers of drones, deliveries and availability windows. While acceptable for the current scope and size in the ILP coursework, under heavier workloads it could become a serious bottleneck. Introducing early filtering stages and reducing repeated iteration over collections would improve efficiency.

Testability:

Due to its size and tightly coupled logic, the function is difficult to unit test. Many behaviours can only be tested through integration tests. Splitting the function into multiple smaller methods would improve testability and allow more focused unit testing.

Overall, while the code is functionally correct and handles all constraints, it lacks in structure, readability, maintainability, and testability. These issues could be addressed by splitting the function into smaller helper methods, and more frequent commenting.