SE2030
Software Engineering
Year 2 – Semester 1

Lab Report – 06

Web based Blood Donation System

Group 1.1 - (MLB-B1G1-03)

Submitted to

Sri Lanka Institute of Information Technology

Student Name	Student ID
Samaraweera P.A.P. B	IT24103858
Manage M.D.N.S.	IT24104013
Wijayagunawardana P.M.P.	IT24103902
Subasinghe I.N.	IT24103980
Gunarathna H.M.S. S	IT24103989
Dilanka P.H.	IT24103939

Design Patterns Used

Our Blood Bank Management System project incorporates the following three design patterns:

1. Singleton Pattern (Creational)
2. Factory Pattern (Creational)
3. Strategy Pattern (Behavioral)

Justifications

A. Singleton Pattern

The Singleton pattern was implemented to manage the application's database connection. The primary objective was to ensure that the DatabaseConnection class would have only one instance throughout the application's lifecycle. This is critical for resource management, as creating multiple database connection objects is inefficient and can lead to inconsistent state management and potential data integrity issues.

By providing a single, global point of access (getInstance()), the Singleton pattern guarantees that all components of the system utilize the same connection object. This centralizes control over the database resource, promoting stability and predictability in data handling operations.

B. Factory Pattern

We utilized the Factory pattern to abstract and centralize the object creation process for BloodRequest entities. The BloodRequestFactory class encapsulates the instantiation logic, thereby decoupling the client (BloodBankApp) from the concrete implementation of the BloodRequest class.

This approach enhances the system's maintainability. If the construction process for BloodRequest objects were to change in the future, for instance, by requiring additional parameters or introducing subclasses—the modifications would be confined to the factory alone. The client code, which requests objects from the factory, would remain unaffected. This separation of concerns makes the codebase cleaner and more adaptable to future changes.

C. Strategy Pattern

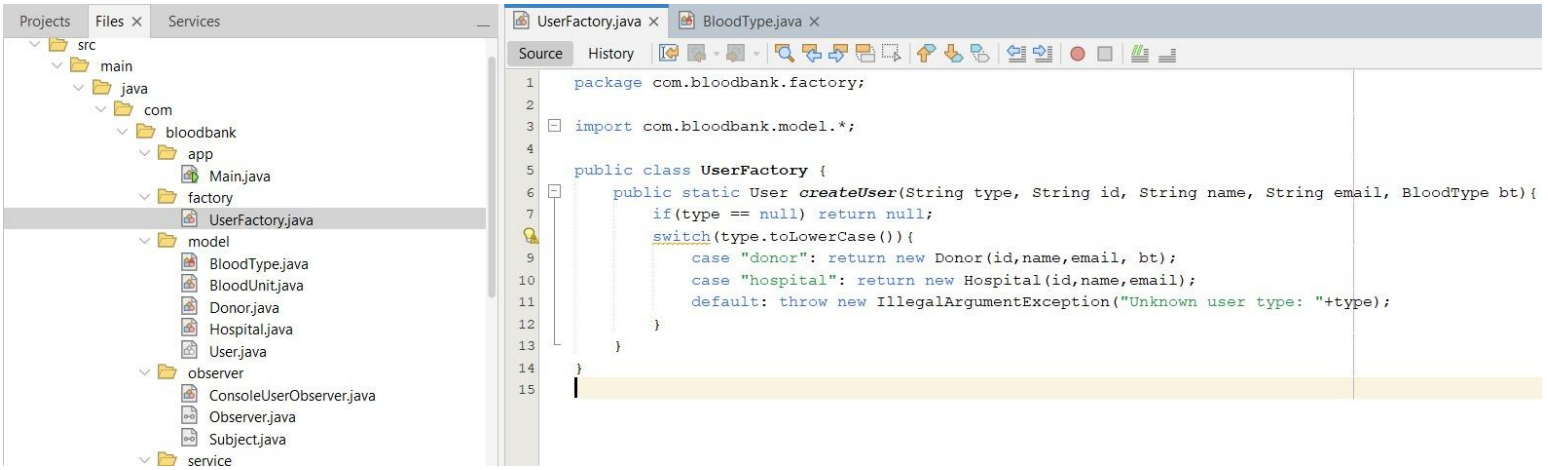
The Strategy pattern was selected to manage the different algorithms for searching for blood donors. The system required multiple search methodologies: a StrictMatchStrategy for finding donors with the exact blood type and a CompatibleMatchStrategy for finding donors based on compatibility rules.

This pattern allows us to encapsulate each algorithm into a separate class, making them

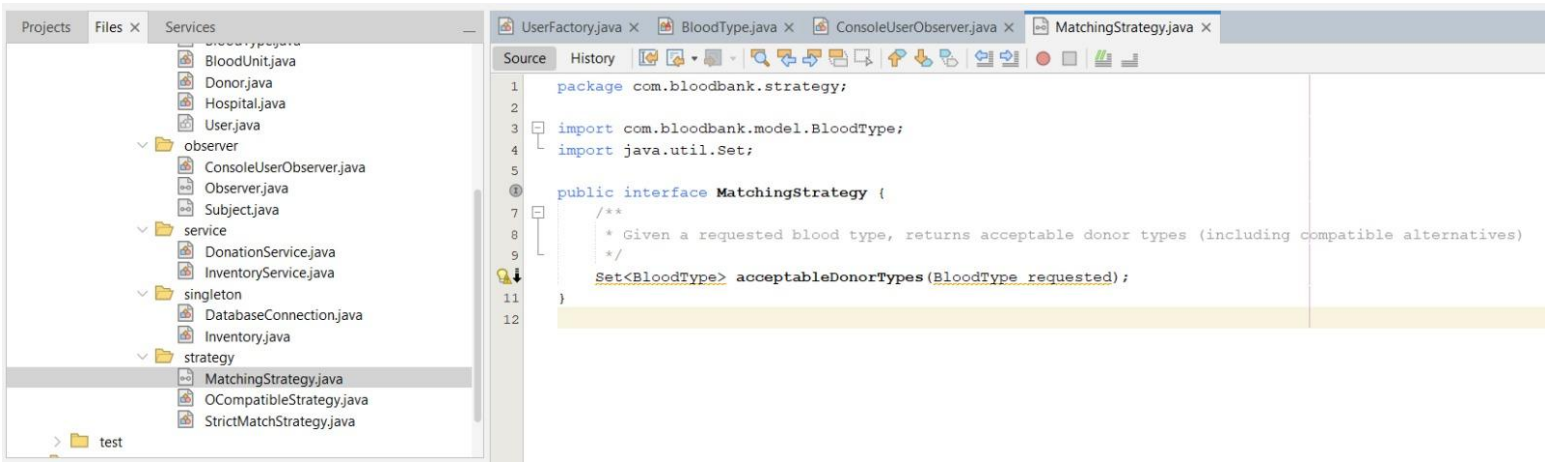
interchangeable at runtime. The BloodBank class can be configured with any SearchStrategy object, delegating the search task to it without being coupled to the specific implementation of the algorithm. This design avoids conditional logic (e.g., if-else blocks) for selecting the search method, adhering to the Open/Closed Principle. As a result, the system is more flexible, and new search strategies can be introduced with minimal modification to the existing codebase.

Screenshots

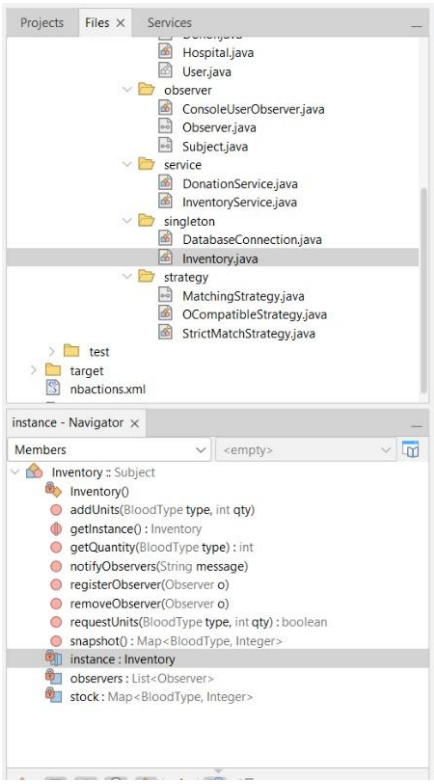
Factory pattern

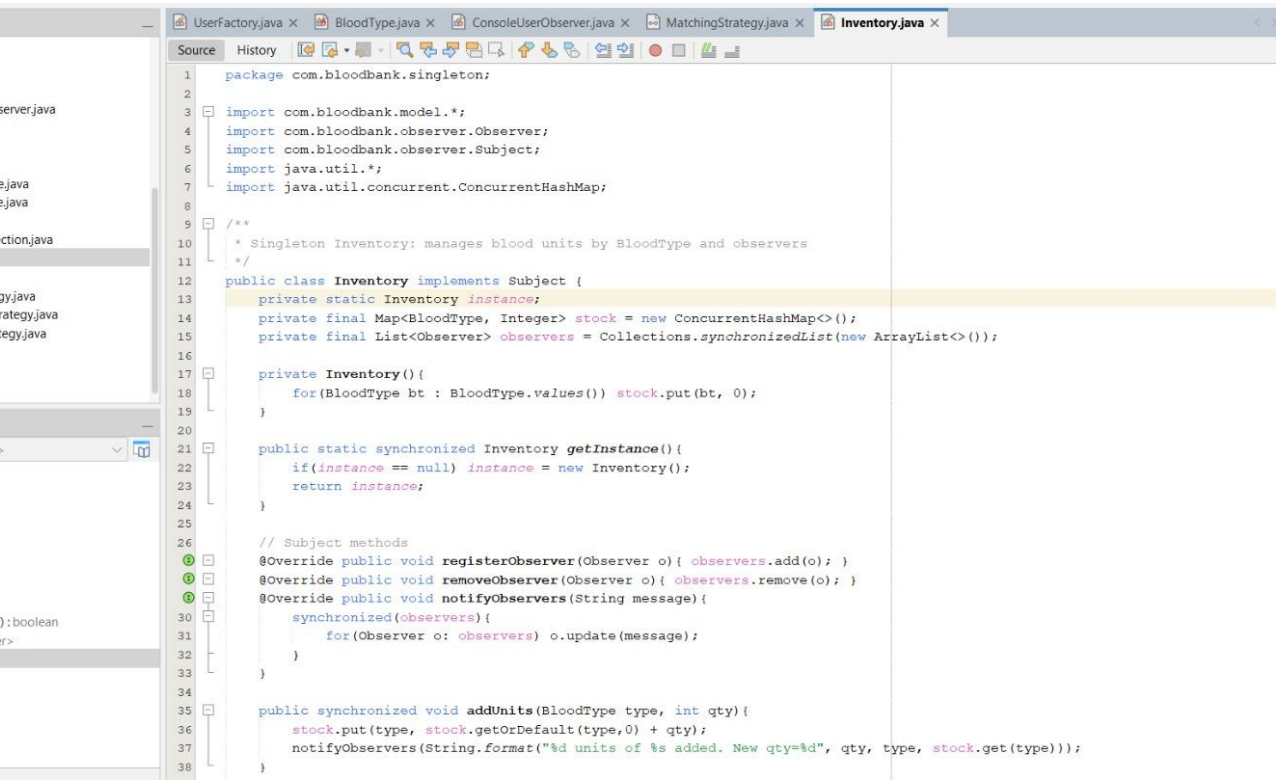


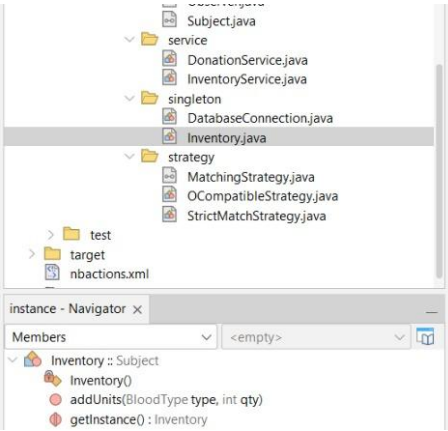
Strategy pattern



Singleton pattern







Console output