**Section 1: Features of the Program**

The program is a web-based notes management application, allowing users to create, view, edit, search, and sort notes. It is implemented using Java and the Jakarta Servlet API, providing functionality to interact with notes stored in a JSON file. The key features of the program are as follows:

1. **Create Notes**: Users can add new notes with a title and content. These notes are stored in a JSON file.
2. **View Notes**: The application displays a list of all stored notes with clickable titles, allowing users to view the content of individual notes.
3. **Edit Notes**: Users can edit the title and content of existing notes. Changes are reflected in the JSON file after submission.
4. **Search**: A search function allows users to filter notes based on a search query, which can match the title or content of a note.
5. **Sort Notes**: The notes can be sorted based on various criteria, including title (ascending/descending) or content size (shortest to longest and vice versa).
6. **UI/UX**: A clean and simple interface is provided for managing notes, featuring a responsive design with easy-to-use forms for adding and editing notes.

**Section 2: Design and Programming Process**

**Design**

The program follows an object-oriented design approach. The key component of the design is the Note class, which encapsulates the properties of a note, such as title, content, and id. This class is responsible for the data structure of a note and provides getter and setter methods for its attributes.

The Model class is designed to handle the logic of storing and retrieving notes. It is implemented as a singleton (using Model.getInstance()) to ensure a single point of access for managing the notes data. This class is responsible for performing CRUD (Create, Read, Update, Delete) operations on the notes and interacting with the JSON file to ensure that notes are updated automatically.

The ModelFactory class acts as a factory for retrieving the model instance, ensuring loose coupling between the servlets and the data layer. This separation of concerns makes it easier to modify the data handling logic in the future without changing the servlet code.

The servlets (ViewNotesServlet, AddNoteServlet, EditNoteServlet, DeleteNoteServlet) handle HTTP requests and responses, implementing the logic for displaying, adding, editing, and deleting notes. Each servlet is designed to handle one specific functionality, adhering to the Single Responsibility Principle of object-oriented design.

**Evaluation of Design**

The design employs several object-oriented principles such as abstraction, encapsulation, and modularity. Each class has a clear responsibility, making the code easy to understand and extend. For example, the Model class abstracts away the complexities of interacting with the storage (JSON file), while the servlets handle user input and delegate tasks to the model.

**Programming Process**

The development was iterative, starting with the basic functionality of displaying and creating notes, followed by implementing the search and sorting features. Each step of development was tested before moving on to the next functionality. The servlets were developed one at a time to handle specific HTTP requests, ensuring that each part of the application was isolated and easy to test.

One challenge encountered was ensuring that the Model class correctly persisted updates to the notes and reflected those changes in the user interface. This was solved by consistently saving and loading the JSON file whenever notes were added, edited, or deleted. The sorting and filtering logic also required careful attention to ensure that the user could sort or filter notes according to their preferences.

**Reflection**

Overall, the program adheres to solid object-oriented design principles, with well-defined responsibilities for each class. The architecture ensures maintainability, with a clear separation of concerns between the model, servlets, and user interface. While the primary focus of the project was to implement a functional web application using object-oriented practices, the user interface remains functional but could benefit from further visual enhancements. Utilizing header.jsp for consistent page headers and styles.css for refined styling would elevate the visual design and improve the overall user experience.