Programming Assignment: Java Port Scanner with GUI

What is port scanner: A port scanner is an application designed to search a network or system for open ports. A port scanner systematically scans a set of port numbers for a given IP address to identify which ports are open and may be accessible. This information is valuable for network administrators to assess the security of a system, as open ports could potentially be exploited by malicious actors if not properly secured. Port scanners are used for both legitimate network management and security testing purposes.

General Objective: Implement a port scanner in Java with a Java FX GUI for user-friendly interaction. The application should allow users to specify an IPv4 address, select a range of ports, initiate the scan, display the results in the GUI, and save the results to a human-readable file.

Minimal features:

Enter an IPv4 address to scan: Develop a user-friendly mechanism within a JavaFX interface to input IPv4 addresses, ensuring validation checks and providing constructive feedback to the user.

Select a Set of Ports to scan: To select a set of port to scan we foresee three different ways:

1. **Enter a Sequence of Comma-Separated Ports:** Users can manually enter a sequence of ports as a comma-separated list in a text field. For example, "80, 443, 8080".
2. **Select a Predefined Set of Ports:** Provide users with predefined sets of ports for common use cases. Here are a few examples of such sets:
    i. Gaming Ports: e.g., 27015, 5000-5010.
    ii. Network-Protocol Ports: e.g., 20 (FTP data), 21 (FTP control), 22 (SSH), etc.
    iii. Webserver Ports: e.g., 80 (HTTP), 443 (HTTPS).
    iv. Common Ports: e.g., 1-1024 (well-known ports).

These predefined sets should be loaded dynamically from a MySQL database. Implement this functionality to load predefined sets of ports from MySQL into the same text field used for manual port entry. Ensure that the format (comma-separated) remains consistent for both predefined and manually entered sets.

3. **Port-Range (Low to High):** Finally, users can opt to specify a port range by entering the starting and ending port numbers, handy for large ranges.

Perform the scan: A simple hit of a button should start the scan.

Show results: Present the scan results to the user in a clear and informative manner. Provide intermediate results during the scan process, indicating the status of each port as either pending, confirmed open, or confirmed closed. Additionally, implement visual cues to convey whether the scan is currently in progress or has been completed in its entirety.

Save the results: Facilitate the seamless saving of scan results to files in a human-readable format upon completion of a scan. Consider implementing a (semi-)automatic saving feature for enhanced user convenience.

Network Scanning Considerations and Ethical Guidelines

- **Network Restrictions:**
  - Be aware that port scanning might be prohibited or restricted on certain networks.
  - Adhere to Howest-network rules and policies during any testing or exploration activities.
- **Recommended Scope:**
  - For "probeersels" (exploratory testing), consider restricting scanning activities to 127.0.0.1 (localhost) and the designated honeyport servers.
  - This approach aligns with ethical considerations and network compliance.

Extra features:

1. **Edit or create a Predefined Set of Ports:** Users can load predefined sets of ports from MySQL. Now, allow users to make manual changes to the loaded predefined sets directly within the text field. Facilitate the addition or removal of ports, providing a unified interface for both predefined and manually entered sets. Upon making manual modifications to a loaded predefined set, prompt the user to decide whether to update the existing set in the database or create a new predefined set. Implement a similar prompt and update/create functionality for manual entry of port sets, allowing users to seamlessly interact with both predefined and manually entered sets. TLDR: The field to

enter a set of ports manually, should also be the input control for inserting and updating these sets into the database.

2. **Load an old scan report:** Add a feature where users can load an old scan report (see 'Save Results') and show the old data again. Now, add another feature that allows the same scan (same ip, same set of ports) again. Find a user friendly way of showing/comparing the old and new results.
3. Note that "uitprobering" your port scanner might be illegal in some cases (see Network Scanning Considerations and Ethical Guidelines). Explore the possibility of creating real unit tests for the port scanner features, recognizing that the complexity of these tests falls outside the scope of the course.

Non-functional Requirements: Clear Separation of Concerns (UI, Data, Service, and Domain)

The most important part of this **exercise**, it to ensure a well-defined separation of concerns within the port scanner application, maintaining clear distinctions between the User Interface (UI), Data layer, Service layer, and Domain logic. This separation aims to enhance maintainability, modularity, and scalability.

Hint: In most applications network-communication is part of the data (or network layer). Load all users from the cloud service, for instance. Pay special thoughts to where you want to put network communication in this app.

Requirements:

1. **User Interface (UI):**
    a. **Responsibility:** The UI layer is responsible for presenting information to the user and receiving user input.
    b. **Requirement:** Implement a modular and intuitive UI design that isolates user interaction from underlying data and business logic.
2. **Data(/Network) Layer:**
    a. **Responsibility:** The Data layer manages the storage, retrieval, and manipulation of data.
    b. **Requirement:** Design a robust data layer that interacts with the database/filesystem for tasks such as loading predefined sets, saving scan results, and managing historical reports.
3. **Service Layer:**
    a. **Responsibility:** The Service layer contains business logic and acts as an intermediary between the UI and the Domain layer.

    b. **Requirement:** Develop a service layer that encapsulates application logic, including loading predefined sets, performing port scans, and managing historical reports.
4. **Domain Layer:**
    a. **Responsibility:** The Domain layer encapsulates the core business logic and entities of the application.
    b. **Requirement:** Ensure that the domain layer remains independent of both UI and data concerns, focusing solely on the business rules related to port scanning, predefined sets, and historical reports.
5. **Inter-Layer Communication:**
    a. **Requirement:** Implement clear and well-defined interfaces for communication between the UI, Data, Service, and Domain layers.
    b. **Requirement:** Utilize appropriate design patterns (e.g., MVC, MVVM) to facilitate modular and loosely coupled interactions between layers.
6. **Modularity and Extensibility:**
    a. **Requirement:** Design the application with a modular structure to allow for ease of maintenance and future extensions.
    b. **Requirement:** Ensure that changes in one layer do not have a cascading impact on other layers, promoting flexibility and extensibility.

Collaborative Approach for Application Development

We promote collaboration in the design phase while advocating in favor of individual efforts in the Java-technical implementation for the port scanner application. Utilize shared platforms for collaborative design discussions, such as video conferencing and collaborative whiteboards, to ideate on UI, architecture, and algorithms. Share FXML wireframes via platforms like Pastebin, draw a comprehensive class diagram together, and collaboratively define SQL code for database design. Develop a protocol for the human-readable file format collaboratively. While jointly designing, emphasize individual responsibility for Java-technical implementation tasks. Schedule regular review meetings for updates and discussions.