Documentation for `AddDrugForm` Class

1. Introduction

This document details the implementation of the `AddDrugForm` class, focusing on its functionalities and interactions with underlying data structures and algorithms.

2. Class Overview

The `AddDrugForm` class is a Java Swing `JPanel` designed for adding new drugs to a pharmacy management system. It utilizes a `Queue<Drug>` to temporarily hold drug objects before inserting them into a database.

3. Implementation Details

3.1 Initialization

The class initializes UI components and layout using `GridBagLayout` through `initComponents()`. It manages drug data using `drugQueue`, which is initialized as a `LinkedList`.

3.2 Event Handling

The `btnSaveActionPerformed` method manages the "Add Drug" button:

1. Retrieves input values.
2. Validates inputs.
3. Creates a `Drug` object and adds it to `drugQueue`.
4. Processes the queue by inserting drugs into the database using `DatabaseHelper.insertDrug`.
5. Displays a confirmation message to the user.

3.3 Drug Class

The `Drug` class represents drug entities with attributes like code, name, price, quantity, supplier, date added, and location.

4. Generating Reports

4.1 Data Structure Implementation Report

- **Data Structure**: `Queue<Drug>`
- **Purpose**: Temporarily holds drugs before database insertion.
- **Operation**: Ensures FIFO order; supports efficient insertion and management.

4.2 Viewing Reports

Users interact with the UI to view confirmation messages and manage drug data updates, ensuring a responsive and informative user experience.

5. Performance Analysis

5.1 Big O Notation

- **Adding a drug to the queue**: O(1)

- **Processing the queue**: O(n) (n is the number of drugs in the queue)

5.2 Omega Notation

- **Best-case performance for adding**: Ω(1)
- **Best-case performance for processing**: Ω(n)

Documentation for `updateTable` Method

1. Introduction

This document describes the `updateTable` method, which updates a JTable with drug data fetched from a database in a Java Swing application.

2. Method Overview

The `updateTable` method clears existing rows in the JTable, fetches updated drug data, and populates the table with rows for each drug entry, including a delete button for user interaction.

3. Implementation Details

3.1 Initialization

- **DefaultTableModel**: Manages table data.
- **JTable**: Displays drug data.
- **Map<String, Drug> drugs**: Stores drug data fetched from the database.

3.2 Data Fetching

- Uses `DatabaseHelper.getAllDrugs()` to fetch drug data into `drugs`.

3.3 Row Addition

- Iterates over `drugs` and adds rows to the table model.
- Each row includes a delete button for managing drug entries.

4. Generating Reports

4.1 Data Structure Implementation Report

- **Data Structure**: `Map<String, Drug>`
- **Purpose**: Efficiently stores and retrieves drug data using drug codes.
- **Operation**: Supports quick access and management of drug entries.

4.2 Viewing Reports

Users view updated drug data directly in the JTable, facilitating effective drug management through clear and organized data presentation.

5. Algorithm Performance Analysis

5.1 Big O Notation

- **Fetching drug data**: O(n) (n is the number of drugs).
- **Clearing table model**: O(1).
- **Iterating and adding rows**: O(n).

5.2 Omega Notation

- **Best-case performance for fetching**: Ω(n).
- **Best-case performance for clearing**: Ω(1).
- **Best-case performance for iterating and adding rows**: Ω(n).

Documentation for `searchDrug` Method

1. Introduction

This document analyzes the `searchDrug` method, focusing on its implementation, data structure usage, and performance metrics.

2. Method Overview

The `searchDrug` method searches for drugs in a pharmacy management system based on a given search term. It updates a JTable to display matching drugs and allows deletion of drug entries.

3. Data Structure Implementation Report

- **Data Structure**: `Map<String, Drug>`
- **Purpose**: Stores drug data for efficient retrieval by drug code.
- **Operation**: Iterates through map entries to find drugs matching the search term.

4. Viewing Reports

Users interact with the UI to view search results directly in the JTable, with feedback provided through message dialogs if no matches are found.

5. Performance Analysis

5.1 Big O Notation

- **Search operation**: O(n) (n is the number of entries).
- **Helper methods**: `containsIgnoreCase`: O(m) (m is the length of the string); `containsDouble` and `containsInteger`: O(1).

5.2 Omega Notation

- **Best-case performance for searching**: Ω(1).
- **Best-case performance for helper methods**: Ω(1).

6. Conclusion

The `searchDrug` method offers efficient drug searching and management within a pharmacy management system, utilizing a map data structure for optimal performance and user interaction.

Documentation for `deleteDrug` Method

1. Introduction

This document outlines the `deleteDrug` method, which removes a drug from a pharmacy management system based on its code.

2. Method Overview

The `deleteDrug` method removes a specific drug entry from both the in-memory data structure and the database, ensuring data consistency.

3. Implementation Details

- **Data Retrieval**: Retrieves all drugs from the database using `DatabaseHelper.getAllDrugs()`.
- **Deletion Process**: Removes the drug from the in-memory map (`Map<String, Drug> drugs`) and calls `DatabaseHelper.deleteDrug(code)` to remove it from the database.
- **UI Update**: Refreshes the UI using `updateTable()` to reflect the deletion.

4. Data Structure

- **Type**: `Map<String, Drug>`
- **Purpose**: Efficiently stores drugs with codes as keys for quick access and modification.

5. Performance Analysis

- **Big O Notation**:
  - Fetching drugs: O(n)
  - Removing from map: O(1)
  - Database deletion: O(1)
- **Omega Notation**:
  - Best-case performance: Ω(1) if the drug exists and deletion is successful.

6. Conclusion

The `deleteDrug` method ensures seamless removal of drugs from the system, maintaining data integrity and providing responsive user interaction.

Documentation for `MergeSort` Class

1. Introduction

This document describes the `MergeSort` class, which provides a static method `mergeSort` to sort a 2D array of Objects based on a specified column index using the mergesort algorithm.

2. Class Overview

The `MergeSort` class implements a sorting algorithm that efficiently sorts a 2D array by a specified column index, ensuring stable and predictable sorting results.

3. Method Overview

- **`mergeSort(Object[][] array, int columnIndex)`**:
  - **Purpose**: Sorts the 2D array of Objects based on the values in the specified column.
  - **Parameters**:
    - `array`: The 2D array of Objects to be sorted.
    - `columnIndex`: The index of the column to sort by.

- **Algorithm**: Uses mergesort to recursively divide the array into halves, sort each half, and merge them back together based on the specified column index.

4. Implementation Details

- **Initialization**:
  - Splits the array into left and right halves.
- **Sorting Process**:
  - Recursively applies mergesort to each half of the array.
  - Merges the sorted halves based on the values in the specified column index.
- **Performance**:
  - **Big O Notation**:
    - Time complexity: O(n log n), where n is the number of elements in the array.
    - Space complexity: O(n), due to auxiliary space used during merging.
  - **Omega Notation**:
    - Best-case time complexity: Ω(n log n), matching the average-case performance.

5. Conclusion

The `MergeSort` class provides an efficient solution for sorting 2D arrays based on a specified column index using mergesort. Its implementation ensures stable performance and clear separation of concerns between sorting logic and data handling in Java Swing applications.