Final Exam Question: Java OOP-Based CRUD Operations Using ArrayLists

Problem Description

You are required to develop a simple inventory management system using Object-Oriented Programming (OOP) principles in Java. The system should allow adding, removing, updating, and searching for items in an inventory. Each item in the inventory will be represented as an object, and you will store these objects in an **`` `ArrayList` ``**. Additionally, you will implement advanced search capabilities and handle specific exceptions using custom exception classes.

Requirements

1. **Item Class**
    - Create a class named **`` `Item` ``** with the following attributes:
        - **`` `id` ``** (String): A unique identifier for each item.
        - **`` `name` ``** (String): The name of the item.
        - **`` `quantity` ``** (int): The quantity of the item available in the inventory.
        - **`` `price` ``** (double): The price of the item.
    - Implement the following methods in the **`` `Item` ``** class:
        - **`` `public Item(String id, String name, int quantity, double price)` ``**: Constructor to initialize all attributes.
        - **`` `public String getId()` ``**: Returns the **`` `id` ``** of the item.
        - **`` `public String getName()` ``**: Returns the **`` `name` ``** of the item.
        - **`` `public int getQuantity()` ``**: Returns the **`` `quantity` ``** of the item.
        - **`` `public double getPrice()` ``**: Returns the **`` `price` ``** of the item.
        - **`` `public void setQuantity(int quantity)` ``**: Sets the **`` `quantity` ``** of the item.
        - **`` `public void setPrice(double price)` ``**: Sets the **`` `price` ``** of the item.
        - **`` `@Override public String toString()` ``**: Returns a string representation of the item, including all attributes.

2. **Inventory Management**

   - Create a class named **`` `Inventory` ``** that manages a collection of **`` `Item` ``** objects using an **`` `ArrayList<Item>` ``**.
   - Implement the following methods in the **`` `Inventory` ``** class:
     - **`` `public void addItem(Item item)` ``**: Adds a new item to the inventory. Ensure that the item with the same **`` `id` ``** does not already exist in the inventory.
     - **`` `public void removeItem(String id) throws ItemNotFoundException` ``**: Removes an item from the inventory based on its **`` `id` ``**. If the item does not exist, throw a custom exception **`` `ItemNotFoundException` ``**.
     - **`` `public void updateItem(String id, int newQuantity, double newPrice) throws ItemNotFoundException` ``**: Updates the quantity and price of an item based on its **`` `id` ``**. If the item does not exist, throw a custom exception **`` `ItemNotFoundException` ``**.
     - **`` `public Item searchItemById(String id) throws ItemNotFoundException` ``**: Searches for an item by its **`` `id` ``** and returns it. If the item is not found, throw a custom exception **`` `ItemNotFoundException` ``**.
     - **`` `public List<Item> searchItemsByName(String name)` ``**: Returns a list of items that match the given **`` `name` ``** (partial matches should be included).
     - **`` `public List<Item> searchItemsByPriceRange(double minPrice, double maxPrice)` ``**: Returns a list of items whose price falls within the specified range.
     - **`` `public List<Item> searchItemsByQuantity(int minQuantity, int maxQuantity)` ``**: Returns a list of items whose quantity falls within the specified range.
     - **`` `public void displayAllItems()` ``**: Displays all items currently in the inventory, formatted in a user-friendly way.

3. **Custom Exception Classes**
   - Create two custom exception classes:
     - **`ItemNotFoundException`**: Thrown when an item with a specified **`id`** is not found in the inventory.
     - **`DuplicateItemException`**: Thrown when trying to add an item with an **`id`** that already exists in the inventory.
   - Each exception should have at least one constructor that takes a **`String`** message and passes it to the superclass **`Exception`**.
4. **Main Application**
   - Create a **`Main`** class with a **`main`** method to test all the functionalities of the **`Inventory`** class.
   - Demonstrate adding, removing, updating, and searching for items.
   - Demonstrate handling of the custom exceptions by providing invalid inputs (e.g., searching for an item that doesn't exist).

Example Scenario

1. Add an item with **`id = "A001"`**, **`name = "Laptop"`**, **`quantity = 10`**, **`price = 999.99`**.
2. Add another item with **`id = "A002"`**, **`name = "Mouse"`**, **`quantity = 50`**, **`price = 19.99`**.
3. Try to add another item with **`id = "A001"`** (should throw **`DuplicateItemException`**).
4. Update the quantity of the item with **`id = "A001"`** to 15 and price to 950.99.
5. Remove the item with **`id = "A003"`** (should throw **`ItemNotFoundException`**).
6. Search for an item by **`id = "A002"`** and display its details.
7. Search for items with a price range between 10.00 and 100.00 and display the results.
8. Search for items with a quantity range between 5 and 20 and display the results.
9. Display all items in the inventory.