Step 1: Setting Up the Basic Structure

- **Goal:** Create the HTML structure for the inventory system.
- **Tasks:**
  - Add a form with input fields for product name and quantity, along with a submit button.
  - Create a container (`div`) to display the list of products.
  - Add a section to display the total quantity of all products.
- **Practice:**
  - Create basic HTML with IDs: `productForm`, `productName`, `productQuantity`, `productList`, and `totalQuantity`.

Step 2: Adding a New Product

- **Goal:** Implement functionality to add a new product to the inventory.
- **Tasks:**
  - Write the `addProduct` method to create a new product with a unique ID and specified quantity.
  - Add the new product to the `products` array and re-render the product list.
  - Reset the form after adding a product.
- **Practice:**
  - Test adding multiple products and verify their appearance in the list.

Step 3: Rendering the Product List

- **Goal:** Display the list of products dynamically in the DOM.
- **Tasks:**
  - Write the `renderProducts` method to loop through the `products` array and create a `div` for each product.
  - Include product details (name and quantity) and buttons for updating quantity and removing the product.
- **Practice:**
  - Style the product list for better appearance using CSS.
  - Test rendering with multiple products.

Step 4: Updating Product Quantity

- **Goal:** Implement functionality to update the quantity of a specific product.

- **Tasks:**
  - Write the `updateProductQuantity` method to find the product by its ID and update its quantity.
  - Add a `promptUpdateQuantity` method to get the new quantity from the user.
  - Validate the input to ensure it's a non-negative number.
- **Practice:**
  - Test updating quantities for different products.

Step 5: Removing a Product

- **Goal:** Allow users to remove a product from the inventory.
- **Tasks:**
  - Write the `removeProduct` method to filter out the product by its ID from the `products` array.
  - Re-render the product list after removing a product.
- **Practice:**
  - Test removing products and verify that the list updates correctly.

Step 6: Calculating Total Quantity

- **Goal:** Calculate and display the total quantity of all products.
- **Tasks:**
  - Write the `calculateTotalQuantity` method to sum up the quantities of all products.
  - Update the `renderProducts` method to display the total quantity in the UI.
- **Practice:**
  - Test with multiple products to verify that the total quantity is accurate.

Step 7: Sorting Products by Name

- **Goal:** Sort the products alphabetically by their name.
- **Tasks:**
  - Write the `sortProductsByName` method to sort the `products` array.
  - Re-render the product list after sorting.
- **Practice:**
  - Test sorting with a variety of product names.

Step 8: Resetting the Form

- **Goal:** Clear the input fields after adding or updating a product.
- **Tasks:**
  - Write the `resetForm` method to reset all input fields.
  - Ensure the form resets after adding or updating a product.
- **Practice:**
  - Test the form with multiple inputs to ensure it resets correctly.

Step 9: Polishing the User Interface

- **Goal:** Improve the visual design and usability of the application.
- **Tasks:**
  - Style the product list, buttons, and form for a professional appearance.
  - Use responsive design techniques to ensure usability on different devices.
- **Practice:**
  - Test the application on various screen sizes and devices.

Bonus Steps

1. **Add Local Storage:**
   - Save the `products` array to `localStorage` to persist data between sessions.
   - Load the data from `localStorage` when the application starts.
2. **Add Search Functionality:**
   - Allow users to search for products by name.
   - Filter the product list dynamically based on the search query.
3. **Add Pagination:**
   - If the product list grows, implement pagination to display products in chunks.
4. **Dark Mode:**
   - Add a toggle for switching between light and dark themes.

Practicing Each Step Focus on implementing and testing each step independently. Once all steps are complete, integrate them to create a fully functional **Inventory Management System**.