System Requirements Specification Index

For

E-Commerce Warehouse Console

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

Table of Contents

E-Commerce Warehouse Console
System Requirements Specification

1 Project Abstract

The **E-Commerce Warehouse Console Application** is a pure Java-based application leveraging Java Collections, Lambda Expressions, Stream API, and Method References to enable seamless management of products in a warehouse. This application provides users with the capability to perform CRUD (Create, Read, Update, Delete) operations and execute essential inventory management functionalities.

The Warehouse Management System empowers users to create new product entries, update product details, delete products, and retrieve products based on specific criteria like price. Additionally, the system calculates the total inventory value.

2 Business Requirements:

Screen Name	Console input screen
Problem Statement	1. User needs to enter into the application. 2. The user should be able to do the below particular operations. 3. The console should display the menu: 1. Add Product 2. Get Products Sorted by Price 3. Get Products Below Specific Price 4. Update Product 5. Delete Product 6. Calculate Total Inventory Value 7. Exit

2.1 InventoryManagementApp Class - Method Descriptions

Method	Task	Implementation Details	Return Value
main(String[] args)	Entry point for the inventory management application	- Initialize Scanner and InventoryService. - Display a menu with numbered options (Add, Update, Delete, etc.). - Read user choice and call the corresponding method. - Handle exceptions: • InvalidProductDataException • DuplicateProductException • IllegalArgumentException • Generic Exception for unexpected errors.	void (Runs continuously until user selects exit)
addProduct(InventoryService inventoryService, Scanner scanner)	Add a new product to the inventory	- Prompt the user: • "Enter product name:" • "Enter product description:" • "Enter product price:" • "Enter product quantity:" - Call inventoryService.addProduct(name, description, price, quantity). - Handle exceptions: • InvalidProductDataException • DuplicateProductException - Print error: "Error adding product: " + exception message	void (Prints success or error message)
getProductsSortedByPrice(InventoryService inventoryService)	Display products sorted by price	- Print: "Products sorted by price:" - Call inventoryService.getProductsSortedByPrice(). - Loop and print each product.	void (Prints sorted list of products)
getProductsBelowSpecificPrice(InventoryService inventoryService, Scanner scanner)	Display products priced below a specific value	- Prompt the user: "Enter price limit:" - Read price input. - Print: "Products below price {limit}:" - Call inventoryService.getProductsBelowPrice(priceLimit).	void (Prints products below specified price)

		- Loop and print each matching product.	
updateProduct(InventoryService inventoryService, Scanner scanner)	Update an existing product's price and quantity	- Prompt the user: • "Enter product ID to update:" • "Enter new price:" • "Enter new quantity:" - Call inventoryService.updateProduct(id, newPrice, newQuantity). - Handle exceptions: • InvalidProductDataException • IllegalArgumentException - Print error: "Error updating product: " + exception message	void (Prints result of update or error message)
deleteProduct(InventoryService inventoryService, Scanner scanner)	Delete a product from the inventory	- Prompt the user: "Enter product ID to delete:" - Read input into id. - Call inventoryService.deleteProduct(id). - Handle IllegalArgumentException if product not found. - Print: "Error deleting product: " + exception message	void (Prints result of deletion or error message)
calculateTotalInventoryValue(InventoryService inventoryService)	Calculate and display total value of inventory	- Call inventoryService.calculateTotalInventoryValue(). - Print: "Total inventory value: " + calculated value	void (Prints total inventory value)

2.2 InventoryService - Method Descriptions

Method	Task	Implementation Details	Return Value
addProduct(String name, String description, double price, int quantity)	Add a new product to the inventory	- Generate a unique product ID using UUID. - Create a Product object with ID, name, description, price, and quantity. - Validate product using Lambda: • Condition: price > 0 and quantity > 0	`void (Adds a product or throws exception)`

		• If invalid, throw InvalidProductDataException with message: "Invalid product data: Price and quantity must be positive." - Check for duplicate product ID using Stream API: • If ID already exists, throw DuplicateProductException with message: "Duplicate ID detected. Cannot add product." - If valid and not duplicate, add product to inventory list. - Print: "Product added: " + product	
getProductsSortedByPrice()	Get list of products sorted by price (ascending)	- Use Stream API to sort inventory by price using Comparator. - Return the sorted list.	`List<Product> (Sorted list of products)`
getProductsBelowPrice(double priceLimit)	Retrieve products priced below a specific value	- Filter products where price < priceLimit using Stream API. - Map filtered products to new Product instances. - Return the list.	`List<Product> (Filtered list of products)`
updateProduct(String id, double newPrice, int newQuantity)	Update price and quantity of a product	- Validate newPrice > 0 and newQuantity >= 0. - If invalid, throw InvalidProductDataException with message: "Invalid update data: Price must be positive, and quantity cannot be negative." - Use Stream API to locate product by ID. - If product is found, update price and quantity. - Print: "Product updated: " + product - If not found, throw IllegalArgumentException with message: "Product with ID: " + id + " not found."	`void (Updates product or throws exception)`

deleteProduct(String id)	Delete a product from inventory	- Check if product with given ID exists using Stream API. - If not, throw IllegalArgumentException with message: "Product with ID: " + id + " not found." - If found, remove product from list. - Print: "Product deleted with ID: " + id	`void (Deletes product or throws exception)`
calculateTotalInventoryValue()	Calculate total monetary value of all products	- Use Stream API to calculate sum of (price * quantity) for each product. - Return the computed value.	`double (Total inventory value)`

3 Constraints

3.1 Add Product

1. Unique ID for Product:
    - Each product must have a unique ID generated using **UUID**.
2. Validate Product Data:
    - Use **Lambda Expression** to validate the product data.
    - The product's price must be greater than `0`, and the quantity must be greater than `0`.
3. Check for Duplicate Product IDs:
    - Use **Stream API** to check if the product ID already exists in the inventory.
4. Add Product to Inventory:
    - Use a **Method Reference** to add the product to the inventory list.

Common Constraints:

1. When attempting to add a product with invalid data (e.g., negative price or quantity), the `addProduct` method should throw an `InvalidProductDataException` with the message:

   "Invalid product data provided: Price and quantity must be positive.”

2. When attempting to add a product with the same ID already exists in the inventory, the `addProduct` method should throw a `DuplicateProductException` with the message:

"Duplicate ID detected. Cannot add product."

3.2 Update Product

1. Validate Updated Data:
    - Use **Lambda Expression** to validate new data.
    - The updated product's price must be greater than `0`, and the quantity must not be negative.
2. Locate and Update Product:
    - Use the **Stream API** to locate the product in the inventory by its ID.

Common Constraints:

1. When updating a product with invalid data (e.g., negative price or quantity), the method should throw an `InvalidProductDataException` with the message:

    “Invalid update data: Price must be positive, and quantity cannot be negative."

2. When attempting to update a product that does not exist in the inventory, the method should throw an `IllegalArgumentException` with the message:

    "Product with ID: " + id + " not found."

3.3 Delete Product

1. Check Product Exists:
    - Use the **Stream API** to check if the product with the given ID exists in the inventory.

Common Constraints::

1. When attempting to delete a product that does not exist in the inventory, the method should throw an `IllegalArgumentException` with the message:

    "Product with ID: " + id + " not found."

3.4 Other Constraints

1. When attempting to retrieve products with a price limit that is negative, the method should return an empty list.
2. When calculating the total inventory value and there are no products in the inventory, the method should return `0.0`.

4 Template Code Structure

4.1 Package: com.ecommerce

Resources

Class/Interface	Description	Status
InventoryManagementApp.java(class)	**This represents bootstrap class i.e class with Main method, that shall contain all console interaction with the user.**	**Partially implemented**

4.2 Package: com.ecommerce.models

Resources

Class/Interface	Description	Status
Product (class)	• This class contains all the properties of the Product class.	Already implemented.

4.3 Package: com.ecommerce.inventory

Resources

Class/Interface	Description	Status
InventoryService (class)	• This class contains all the methods which are used to write the business logic for the application. • You can create any number of private methods in the class.	Partially implemented.

4.4 Package: com.ecommerce.exception

Resources

Class/Interface	Description	Status
DuplicateProductException (Class)	• Custom Exception to be thrown when trying to add a product that already exists in the inventory, identified by a duplicate product id.	Already created.
InvalidProductDataException (Class)	• Custom Exception to be thrown when trying to add or update a product with invalid data.	Already created.

5 Execution Steps to Follow

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.**
3. **This editor Auto Saves the code.**
4. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
5. **To run your project use command:**
   mvn clean install exec:java -Dexec.mainClass="com.ecommerce.InventoryManagementApp"
7. **To test your project, use the command**
   mvn test