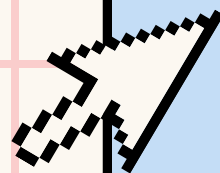




by KariAnn Harjo

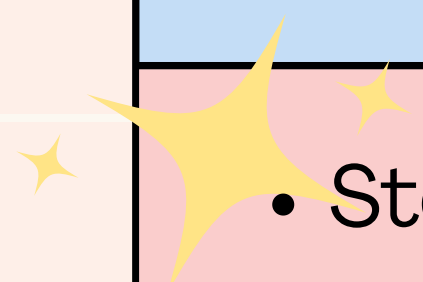

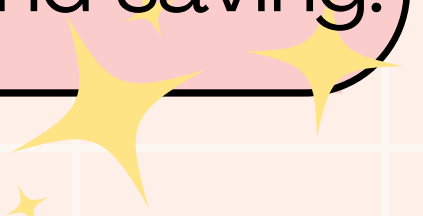
STOREFRONT APPLICATION

#aRobustJavaApplication





APPLICATION FUNCTIONALITY

- 
- StoreFrontApplication facilitates purchase and management of products.
 - Supports purchasing, canceling purchases, viewing inventory, cart management, and checkout functionality.
 - Provides administrative capabilities, enabling product management and inventory updates.
 - Implements multithreading for efficient inventory loading and saving.
- 
- 



FUNCTIONALITY EXAMPLES

A piece of code showcasing the initialization and loading of inventory from a JSON file

```
000 +  
  
// Initialize the store with the initial inventory from the JSON file  
store.getInventoryManager().loadInventoryFromJsonFile("inventory.json");  
  
// Create a shopping cart instance  
ShoppingCart<SalableProduct> cart = new ShoppingCart<>();  
  
System.out.println("Welcome to Bloodbath and Beyond,");  
System.out.println("where you can prep for your next battle or bath!\n");  
System.out.println("Available Actions:");
```

Code representing
the user
interaction to
make purchases

```
System.out.println("Available products:");  
for (SalableProduct product : store.getInventory()) {  
    System.out.println(product.getName());  
}  
System.out.print("Enter the name of the product: ");  
String productName = scanner.nextLine();  
System.out.print("Enter the quantity: ");  
int quantity = scanner.nextInt();  
SalableProduct selectedProduct = store.getInventory().get(productName);  
if (selectedProduct != null) {  
    store.purchaseProduct(selectedProduct, quantity);  
} else {  
    System.out.println("Product not found.");  
}  
break;
```

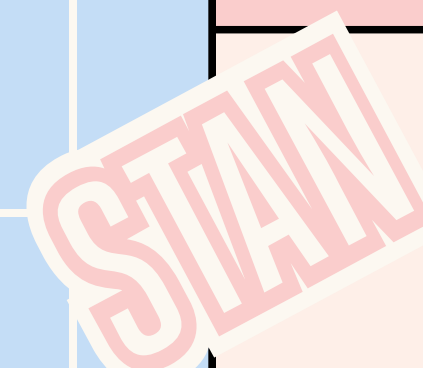
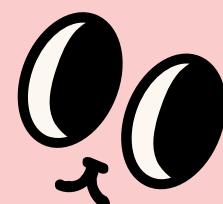
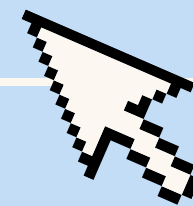
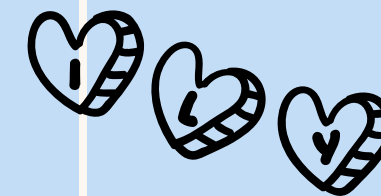


GOALS AND DESIGN CHOICES

- Use of modular, object-oriented design allowing for easy extension and maintenance.
- Implementation of multithreading to handle inventory tasks concurrently.
- Utilization of JSON for inventory data storage and retrieval, ensuring data persistence and interoperability.
- Implementation of separate administration and storefront modules to segregate functionalities and improve security.



Goal: Create a user-friendly, efficient, and extendable storefront and inventory management application.



GOALS AND DESIGN EXAMPLES

Use of object-oriented design:
Defining a SalableProduct class with appropriate attributes and methods.

```
✕ □ —
// Class representing a product that can be sold
public class SalableProduct implements Comparable<SalableProduct> {

    // Member variables holding the product's details
    private String name;
    private String description;
    private double price;
    private int quantity; // Number of items available
    private int quantityPurchased; // Number of items purchased

    // Constructor to initialize member variables using JsonCreator
    // as the objects are created through deserialization of JSON.
    @JsonCreator
    public SalableProduct(@JsonProperty("name") String name,
                          @JsonProperty("description") String descri
                          @JsonProperty("price") double price,
                          @JsonProperty("quantity") int quantity,
                          @JsonProperty("quantityPurchased") int qua

        this.name = name;
        this.description = description;
    }
}
```

Implementation of multithreading for concurrent inventory load

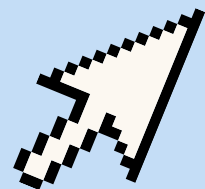
```
○○○ +
public void loadInventoryFromJsonFile(String filePath) {
    Thread loadInventoryThread = new Thread(() -> {
        ObjectMapper objectMapper = new ObjectMapper();
        try {
            // Deserialize JSON data from the file into a list of SalableProdu
            SalableProduct[] loadedProducts = objectMapper.readValue(new File(

            // Clear the existing inventory and add the loaded products
            productList.clear();
            Collections.addAll(productList, loadedProducts); // Use Collection
        } catch (JsonParseException | JsonMappingException e) {
        }
    });
}
```



CHALLENGES ENCOUNTERED

1. Efficiently managing concurrent tasks for inventory load and save operations.
2. Ensuring data integrity and consistency across different modules and threads.
3. Designing a flexible and extendable architecture to accommodate future enhancements.
4. Debugging and resolving complex multithreading and synchronization issues.



CHALLENGES ENCOUNTERED

Handling concurrency issues in the ShoppingCart class using 'synchronized' keyword

```
public class StoreFront {  
    private InventoryManager inventoryManager;  
    private List<ShoppingCart<SalableProduct>>  
  
    public StoreFront() {  
        inventoryManager = new InventoryManager  
    }  
  
    public InventoryManager getInventoryManager  
        return inventoryManager;  
    }  
  
    public void purchaseProduct(SalableProduct  
        if (inventoryManager.getProductList().c
```

Designing a flexible and extendable architecture, Example of Modular Design through various classes like StoreFront, InventoryManager

```
public synchronized void removeProduct(T product, int qu  
    if (cartItems.containsKey(product)) {  
        int currentQuantity = cartItems.get(product);  
        if (currentQuantity > quantity) {  
            cartItems.put(product, currentQuantity - qua  
            totalPrice -= product.getPrice() * quantity;  
            System.out.println("Removed " + quantity + "  
        } else if (currentQuantity == quantity) {  
            cartItems.remove(product);  
            totalPrice -= product.getPrice() * quantity;  
            System.out.println("Removed " + quantity + "  
        } else {  
            System.out.println("Invalid quantity to remo
```

Console Declaration Terminal



PENDING BUGS OR ISSUES

Enhancements:

User Interface Improvements:

Implementing a graphical user interface (GUI) or a web-based user interface would improve user experience and accessibility.

Extensive Error Handling and User Input Validation:

Implement more comprehensive error handling and input validation to manage user inputs effectively. More checks and clearer error messages would enhance the user experience.

Performance Optimizations:

Currently, the application loads and saves the inventory to and from a JSON file for every operation. This is not optimal for performance. A caching mechanism could be implemented to load the inventory once and only write back to the file at certain intervals or under certain conditions.

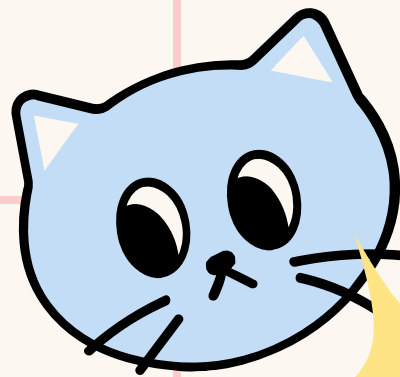
Logging Mechanism:

Implement a proper logging mechanism to keep track of all the actions performed within the application, which would be extremely helpful for debugging and auditing purposes.

Enhanced Sorting and Filtering:

The application could benefit from more advanced sorting and filtering options for the products, for instance, filtering products by categories, brands, price ranges, and sorting by popularity, ratings, etc.

BIAS



KEY LEARNINGS
AND
FUTURE
APPLICATION

○○○	
Online Reality Shows	What to Expect
<u>Modular Design</u>	Emphasizing modularity aids in managing complexity and facilitates future extensions.
<u>Multithreading</u>	Effective use of multithreading can significantly improve application responsiveness and performance.
<u>Data Serialization</u>	Proficient use of serialization techniques like JSON aids in seamless data storage and retrieval.
<u>Effective Testing</u>	Incorporating JUnit tests ensures the reliability of each class and method, aiding in early bug detection.
<u>User Interaction</u>	Creating intuitive and clear user interfaces is crucial for user experience and application usability.

LEARNING EXAMPLES

```
× □ —
    inventoryManager = new InventoryManager(st
    }

    @Test
    public void testAddProduct() {
        SalableProduct product = new SalableProduct
        inventoryManager.addProduct(product);

        // Check whether the added product is present
        assertTrue(inventoryManager.getProductList()
    }

    @Test
    public void testFindProductByName() {
```

Example showing effective testing by utilizing JUnit.

Example showing the implementation of JSON serialization for inventory data storage and retrieval

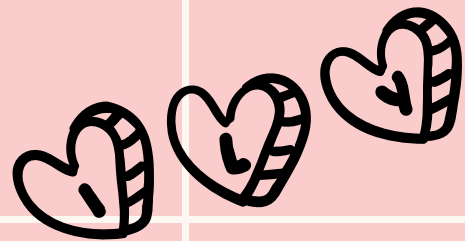
```
— □ ×
    // Add inventory in thread
    ObjectMapper objectMapper = new ObjectMapper();

    try {
        // Deserialize JSON data from the file into
        SalableProduct[] loadedProducts = objectMapper.readValue(
            file, SalableProduct[].class);

        // Clear the existing inventory and add the loaded products
        productList.clear();
        Collections.addAll(productList, loadedProducts);
    } catch (JsonParseException | JsonMappingException e) {
        e.printStackTrace();
    } // Handle JSON parsing errors
    catch (IOException e) {
        e.printStackTrace();
    } // Handle File I/O errors
```



CONCLUSION



StoreFrontApplication
is a comprehensive
solution for storefront
and inventory
management.



It employs advanced
features like
multithreading, JSON
serialization, modular
design, and
comprehensive
testing.

The learnings from this
project will serve as a
valuable foundation for
developing advanced,
robust, and user-
friendly applications in
the future.

