ALBUKHARY INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTING AND INFORMATICS
SEMESTER 2 2023/2024

CCC1223
OBJECT ORIENTED PROGRAMMING
GROUP PROJECT - ASSIGNMENT 3

PROJECT TITLE		
GROUP NO		
GROUP LEADER	**Hasibullah Naeim**	**AIU22102121**
GROUP MEMBERS	**Muhammad Zulfan Abidin** **ALMONZER HAMID SARRAY ABDALLAH** **Mohamed Naseer Mohamed Afrath (GP-E)** **ALTAGI ABDALLAH BAKHEIT ABDELGADIR** **Osamah Hasan Mohammed Alrebaki**	**AIU 22102088** **AIU22102343** **AIU 23102126** **AIU 22102296** **AIU22102105**

For Examiner's use only

ITEMS	**MARKS**
Report (Marks)	

Table of Content

Introduction:

Effective inventory management is essential to any organization's success and long-term viability in the fast-paced, fiercely competitive corporate world of today. A well-functioning inventory management system guarantees that the appropriate amount of goods is accessible at the appropriate moment, reducing expenses and optimising client contentment. The goal of our project, the "Inventory Management System," is to make inventory tracking and management in a commercial environment more efficient.

The goal of this system's design is to offer a complete solution for preserving ideal stock levels, cutting waste, and raising overall operational effectiveness. Our technology will assist firms in avoiding stockouts and overstocking, hence lowering the risk of financial losses, by automating the inventory process. Businesses will be able to use the system's capabilities, which will include automated reorder notifications, real-time inventory management, and comprehensive reporting.

Our idea makes advantage of contemporary technology to provide an intuitive user interface that makes inventory management jobs easier. By putting this system in place, organisations will be able to concentrate on their core competencies and strategic expansion while also improving inventory accuracy and freeing up critical time and resources. The ultimate goal of the "Inventory Management System" is to contribute to the organization's success by offering a strong basis for effective supply chain management.

B. Design:

The Inventory Management System is built using three main classes: Product, Inventory, and InventoryManagementSystem are the labels of 3 different entities. Each class has a specific role in the system:Each class has a specific role in the system:

1. Product Class:

This class gives an abstraction of each unique product in the inventory. It encapsulates the following attributes:It encapsulates the following attributes:
- productId (int): The identification that is required for each product
- productName (String): Manufacturing brand of the retail product
- price (double): The price at which the product was bought
- quantity (int): In particular, it is necessary to analyze the availability of the product on the market and decide whether there is enough of it for the targeted audience.

Product class includes accessor methods for all of the attributes and modifier methods for all of the attributes except for productId. It also implements its own customized version of the toString() method for quick display and prompt storage in a file.

2. Inventory Class:

This class takes responsibility for Product objects and offers methods that relate to inventory handling. Key features include:
- products (Product[]): This is an array in which Product objects can be stored.
- count (int): Measures the quantities actually on hand in terms of the various products in the inventory.

The Inventory class implements the following methods: The Inventory class implements the following methods:
- addProduct(): Cuts: This operation involves adding a new product to the inventory.
- removeProduct(): Take out a product by the unique identification number
- updateProduct(): Used to amend information about a specific product that has already been posted on the website.

- searchProduct(): To find a particular product and bring out its details on the Product Details page
- displayProducts(): Shows all stock available in the store or in other words presents all products in the store.
- saveToFile(): Stores a record of the data stored in the inventory to a file
- loadFromFile(): Reads data from a file and fills the inventory
- expandArray(): Periodically increases the dimension of the products array when necessary

InventoryManagementSystem Class:
This is the main class in the program which is holding the user interface and controlling the flow of the program. It contains:

A static Scanner object for user input Gör<|reserved_special_token_251|>
You create a static Inventory object that helps you track and organize the products.
The method that contains high-level logic of the program and controls the program flow with the help of the loop drop
Here are the static and private methods for each of the menu options

Relationships between Classes:

The class named InventoryManagementSystem works with the Inventory class that is used for storing products.
Simply, the class named Inventory is responsible for storing and controlling an array of objects of the Product class.
The Product class is a subclass of the Serializable class and is employed in the Inventory and InventoryManagementSystem classes to model product objects.

This design follows object-oriented principles: This design follows object-oriented principles:

Encapsulation: Every single class can then store its data and offer functionality for handling it.
Single Responsibility: They are all integral parts of the global system and each is assigned a function.
Modularity: It is a modular system where that can be split according to relatively independent subsystems (classes).

It organizes well the present reality of the project, more in what comes to the menu system, using a multiplicity of classes, application of methods and constructors, an array of objects, and file I/O. They do not change any part of the code to meet the requirements of the project that was pursued.

Implementation

Product Class

The Product class represents an individual product in the inventory. It has the following attributes:

- productId: an integer that uniquely identifies the product.
- productName: a string that holds the name of the product.
- price: a double that holds the price of the product.
- quantity: an integer that holds the quantity of the product available in inventory.

The class includes a constructor for initializing these attributes, getter and setter methods for accessing and modifying them, and a toString method for representing the product as a string.

Inventory Class

The Inventory class manages a collection of Product objects using a List<Product>, specifically an ArrayList.

- **addProduct(Product product)**: Adds a product to the inventory.
- **removeProduct(int productId)**: Removes a product from the inventory based on its productId.
- **updateProduct(Product updatedProduct)**: Updates the details of a product in the inventory. It searches for the product by its productId and updates its attributes if found.
- **searchProduct(int productId)**: Searches for a product in the inventory by its productId. Returns the product if found, or null otherwise.
- **displayProducts()**: Displays all products in the inventory by printing their details to the console.

- **saveToFile(String filename)**: Saves the current list of products to a file. Each product's details are written on a new line in the file.
- **loadFromFile(String filename)**: Loads products from a file into the inventory. Each line in the file is parsed to create a Product object, which is then added to the inventory.

InventoryManagementSystem Class

The InventoryManagementSystem class provides a console-based interface for interacting with the inventory. It uses a Scanner for user input and maintains a static instance of the Inventory class.

- **main(String[] args)**: The main method which runs an infinite loop to display a menu and process user choices.
- **showMenu()**: Displays the menu options to the user.
- **addProduct()**: Prompts the user to enter product details and adds the new product to the inventory.
- **removeProduct()**: Prompts the user to enter a product ID and removes the corresponding product from the inventory.
- **updateProduct()**: Prompts the user to enter product details and updates the corresponding product in the inventory.
- **searchProduct()**: Prompts the user to enter a product ID and searches for the product in the inventory. Displays the product details if found.
- **displayProducts()**: Displays all products in the inventory.
- **saveToFile()**: Prompts the user to enter a filename and saves the current inventory to the file.
- **loadFromFile()**: Prompts the user to enter a filename and loads the inventory from the file.

Challenges and Solution

Array Resizing:

- **Challenge**: Initially, the inventory was implemented using a fixed-size array, which posed challenges when the number of products exceeded the array size.
- **Solution**: Switched to using ArrayList<Product>, which dynamically resizes as needed. This simplified adding products and avoided the complexity of manually resizing arrays.

Error Handling:

- **Challenge**: Ensuring that invalid user inputs (e.g., non-numeric values, negative quantities, etc.) do not cause the program to crash.
- **Solution**: Added input validation and exception handling. For instance, used try-catch blocks to catch InputMismatchException and provided meaningful error messages to guide the user.

File I/O Operations:

- **Challenge**: Managing file read/write operations to ensure data consistency and handling potential I/O errors.
- **Solution**: Used try-with-resources for file operations to ensure resources are closed properly. Added error messages to inform users when file operations fail.

Maintaining Unique Product IDs:

- **Challenge**: Ensuring each product has a unique ID when adding new products.
- **Solution**: Added a check to verify that the product ID does not already exist in the inventory before adding a new product. If it does, prompt the user to enter a different ID.

User Interface and Experience:

- **Challenge**: Creating an intuitive and user-friendly interface for the console-based application.
- **Solution**: Displayed clear and concise menu options. Provided confirmation messages for operations like adding, updating, and removing products. Used prompts to guide the user through the required inputs.

Data Parsing from File:

- **Challenge**: Ensuring that the data format in the file is consistent and correctly parsed when loading products from a file.
- **Solution**: Implemented data validation when reading from the file. Checked for the correct number of data fields and parsed them carefully, handling potential NumberFormatException errors.

Updating Product Details:

- **Challenge**: Efficiently updating product details while ensuring that the correct product is identified.
- **Solution**: Used a loop to find the product by its ID and then updated its attributes. This approach ensures that only the intended product is modified.

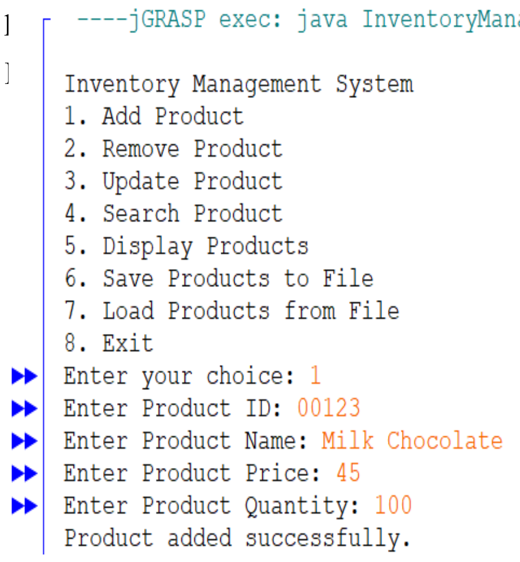
Output

The Inventory Management System is designed with a straightforward and user-friendly interface that facilitates seamless interaction for managing products within a store. Upon launching the program, users are presented with a menu displaying various options. The menu includes commands to add a product, remove a product, update product details, search for a product by its ID, display all products in the inventory, save the current product list to a file, load product data from a file, and exit the application.

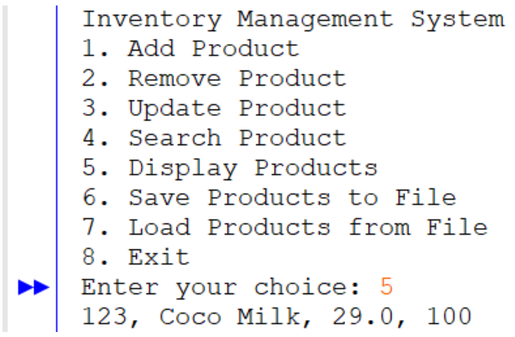
To use the program, the user must input the number corresponding to the desired operation and press Enter. For example, to add a product, the user selects option 1, after which the program prompts for the product's ID, name, price, and quantity. After this, it will display that “**Product added successfully**” automatically. At the same time, if the user selects "Display

] letails as Product ID, Product Name,

]

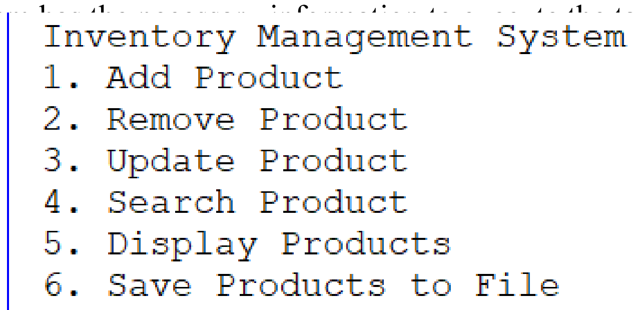


```
----jGRASP exec: java InventoryManagementSystem

Inventory Management System
1. Add Product
2. Remove Product
3. Update Product
4. Search Product
5. Display Products
6. Save Products to File
7. Load Products from File
8. Exit
Enter your choice: 1
Enter Product ID: 00123
Enter Product Name: Milk Chocolate
Enter Product Price: 45
Enter Product Quantity: 100
Product added successfully.
```



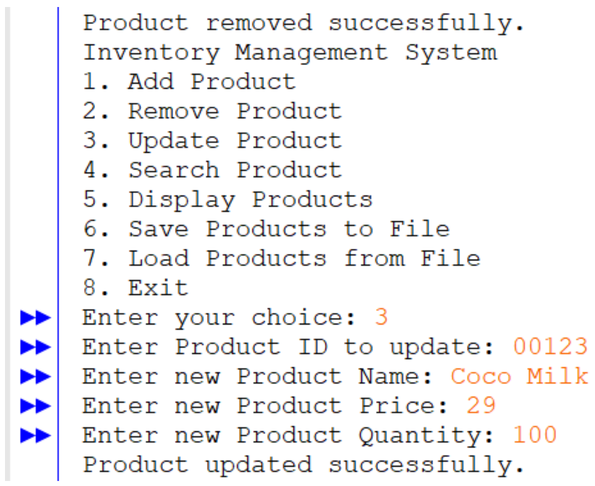
```
Inventory Management System
1. Add Product
2. Remove Product
3. Update Product
4. Search Product
5. Display Products
6. Save Products to File
7. Load Products from File
8. Exit
Enter your choice: 5
123, Coco Milk, 29.0, 100
```

Similarly, selecting option 2 prompts the user to enter the ID of the product they wish to remove. Each action requires the user to provide specific details pertinent to the operation, ensuring the



```
Inventory Management System
1. Add Product
2. Remove Product
3. Update Product
4. Search Product
5. Display Products
6. Save Products to File
```

The system's design also emphasizes simplicity and clarity, guiding users through each step with clear instructions. For instance, when updating a product, the user is asked to enter the product ID, followed by new values for the product's name, price, and quantity. The program handles invalid inputs gracefully, prompting users to re-enter correct information when necessary.

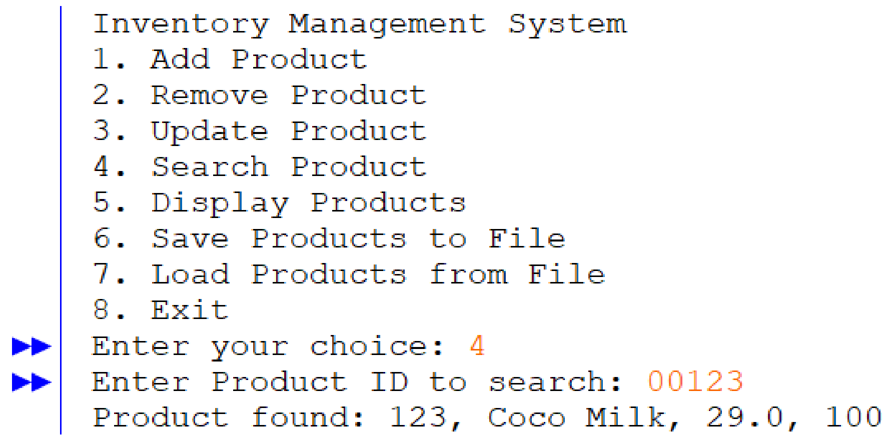


```
Product removed successfully.
Inventory Management System
1. Add Product
2. Remove Product
3. Update Product
4. Search Product
5. Display Products
6. Save Products to File
7. Load Products from File
8. Exit
Enter your choice: 3
Enter Product ID to update: 00123
Enter new Product Name: Coco Milk
Enter new Product Price: 29
Enter new Product Quantity: 100
Product updated successfully.
```

After each operation, the system provides feedback, such as confirming a successful addition or update of a product, ensuring users are aware of the outcome of their actions.

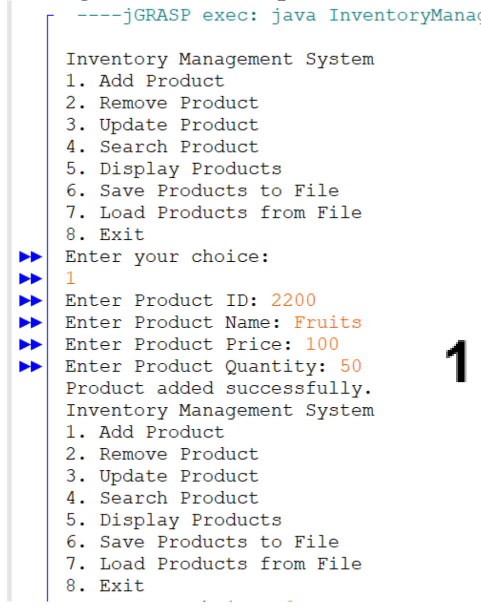
The user interface of the Inventory Management System is console-based, offering a text-driven interaction model. This interface begins with a welcome message followed by a numbered menu displaying the various operations available. Each menu item is clearly labeled with a number and a description, such as "1. Add Product" or "2. Remove Product," making it easy for users to understand and navigate the system. The interface operates in a loop, continuously displaying the menu after each operation until the user chooses to exit by selecting option 8. This design ensures that users can perform multiple operations in a single session without needing to restart the application. The prompt at the end of each operation allows users to enter their next choice immediately, maintaining a smooth and uninterrupted workflow.

When users select an option, the program provides specific prompts relevant to that action. For example, if the user selects "Search Product" (option 4), the interface asks for the product ID and then displays the details of the found product or a message indicating that the product was not found.

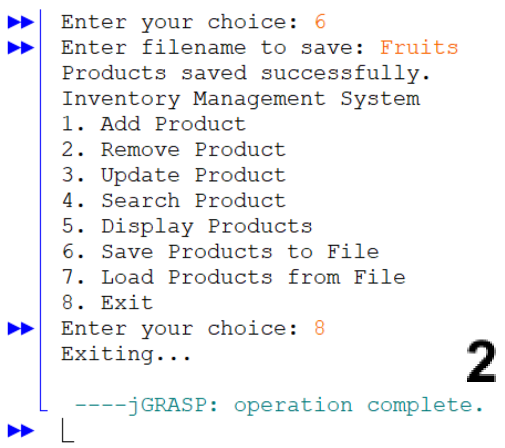


```
Inventory Management System
1. Add Product
2. Remove Product
3. Update Product
4. Search Product
5. Display Products
6. Save Products to File
7. Load Products from File
8. Exit
Enter your choice: 4
Enter Product ID to search: 00123
Product found: 123, Coco Milk, 29.0, 100
```

For file operations, such as saving products, the interface prompts the user to enter the filename, ensuring that the user specifies the correct file path for these actions.

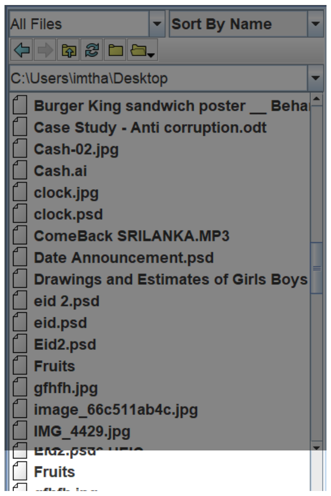


```
----jGRASP exec: java InventoryManagementSystem

Inventory Management System
1. Add Product
2. Remove Product
3. Update Product
4. Search Product
5. Display Products
6. Save Products to File
7. Load Products from File
8. Exit
Enter your choice:
1
Enter Product ID: 2200
Enter Product Name: Fruits
Enter Product Price: 100
Enter Product Quantity: 50
Product added successfully.
Inventory Management System
1. Add Product
2. Remove Product
3. Update Product
4. Search Product
5. Display Products
6. Save Products to File
7. Load Products from File
8. Exit
```

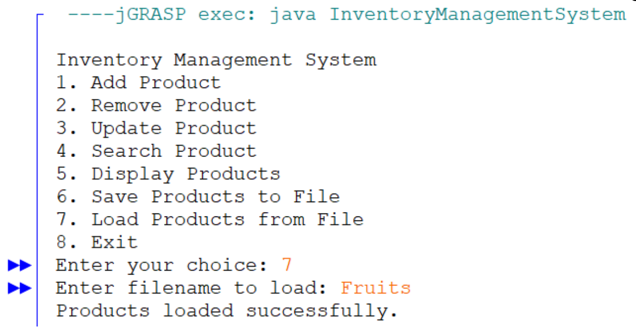


```
Enter your choice: 6
Enter filename to save: Fruits
Products saved successfully.
Inventory Management System
1. Add Product
2. Remove Product
3. Update Product
4. Search Product
5. Display Products
6. Save Products to File
7. Load Products from File
8. Exit
Enter your choice: 8
Exiting...

 ----jGRASP: operation complete.
```

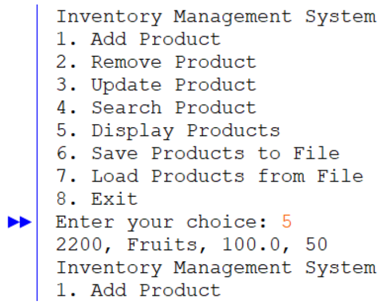
3



When loading products from a file (using option 7), the interface asks the user to enter the filename, ensuring that the user specifies the correct file path for these actions. After that, to display the content of loaded file we should chose option 5 (display products).



```
----jGRASP exec: java InventoryManagementSystem

Inventory Management System
1. Add Product
2. Remove Product
3. Update Product
4. Search Product
5. Display Products
6. Save Products to File
7. Load Products from File
8. Exit
Enter your choice: 7
Enter filename to load: Fruits
Products loaded successfully.
```



```
Inventory Management System
1. Add Product
2. Remove Product
3. Update Product
4. Search Product
5. Display Products
6. Save Products to File
7. Load Products from File
8. Exit
Enter your choice: 5
2200, Fruits, 100.0, 50
Inventory Management System
1. Add Product
```

Overall, the console-based user interface is designed to be intuitive and accessible, requiring no prior technical expertise to operate. The text prompts and feedback messages are crafted to guide users effectively through each operation, making the Inventory Management System a practical tool for managing store inventories.

Conclusion

In conclusion, our Inventory Management System project successfully demonstrates the application of various Object-Oriented Programming (OOP) concepts, such as class design, object arrays, and file handling. By completing this project, we achieved the following:

Practical Application of OOP Concepts:
We designed and implemented a system with multiple classes, showcasing inheritance, encapsulation, and polymorphism. This practical application reinforced our understanding of these fundamental OOP principles.

File Handling:
We implemented methods for saving and loading product data to and from a file, ensuring data persistence. This feature is crucial for real-world applications where data needs to be retained across sessions.

User Interaction:
The system includes a user-friendly menu-driven interface, allowing users to add, remove, update, search, and display products with ease. This interactive aspect is vital for a positive user experience.

Challenges and Solutions:
Throughout the project, we encountered challenges, particularly in file handling and ensuring data integrity during updates and deletions. By collaboratively troubleshooting and testing various approaches, we developed robust solutions to these issues.

Overall, this project not only met the specified requirements but also provided us with invaluable experience in designing, implementing, and testing a real-world application. Moving forward, we see potential for further enhancements, such as incorporating a graphical user interface (GUI) for improved usability and expanding the system's capabilities to handle more complex inventory management tasks.

Source Code:

```
import java.util.Scanner;

class Product {
    private int productId;
    private String productName;
    private double price;
    private int quantity;

    public Product(int productId, String productName, double price, int quantity) {
        this.productId = productId;
        this.productName = productName;
        this.price = price;
        this.quantity = quantity;
    }

    public int getProductId() {
        return productId;
    }

    public String getProductName() {
        return productName;
    }

    public double getPrice() {
        return price;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    @Override
    public String toString() {
        return productId + ", " + productName + ", " + price + ", " + quantity;
    }
}

class Inventory {
    private Product[] products;
    private int count;

    public Inventory() {
        products = new Product[100];  // initial capacity of 100
        count = 0;
    }

    public void addProduct(Product product) {
        if (count >= products.length) {
            expandArray();
        }
```

```
   products[count++] = product;
}

public void removeProduct(int productId) {
   for (int i = 0; i < count; i++) {
      if (products[i].getProductId() == productId) {
         products[i] = products[count - 1];  // replace with the last product
         products[count - 1] = null;  // remove reference for GC
         count--;
         break;
      }
   }
}

public void updateProduct(Product updatedProduct) {
   for (int i = 0; i < count; i++) {
      if (products[i].getProductId() == updatedProduct.getProductId()) {
         products[i].setProductName(updatedProduct.getProductName());
         products[i].setPrice(updatedProduct.getPrice());
         products[i].setQuantity(updatedProduct.getQuantity());
         break;
      }
   }
}

public Product searchProduct(int productId) {
   for (int i = 0; i < count; i++) {
      if (products[i].getProductId() == productId) {
         return products[i];
      }
   }
   return null;
}
```

```
public void displayProducts() {
   for (int i = 0; i < count; i++) {
      System.out.println(products[i]);
   }
}

public void saveToFile(String filename) throws IOException {
   try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
      for (int i = 0; i < count; i++) {
         writer.write(products[i].toString());
         writer.newLine();
      }
   }
}

public void loadFromFile(String filename) throws IOException {
   products = new Product[100];  // reset array
   count = 0;
   try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
      String line;
      while ((line = reader.readLine()) != null) {
         String[] parts = line.split(", ");
         int productId = Integer.parseInt(parts[0]);
         String productName = parts[1];
         double price = Double.parseDouble(parts[2]);
         int quantity = Integer.parseInt(parts[3]);
         addProduct(new Product(productId, productName, price, quantity));
      }
   }
}

private void expandArray() {
        Product[] newProducts = new Product[products.length * 2];
        System.arraycopy(products, 0, newProducts, 0, products.length);
        products = newProducts;
    }
}

public class InventoryManagementSystem {
    private static final Scanner scanner = new Scanner(System.in);
    private static final Inventory inventory = new Inventory();

    public static void main(String[] args) {
        while (true) {
            showMenu();
            int choice = scanner.nextInt();
            scanner.nextLine();  // Consume newline

            switch (choice) {
                case 1:
                    addProduct();
                    break;
                case 2:
                    removeProduct();
                    break;
                case 3:
                    updateProduct();
                    break;
                case 4:
                    searchProduct();
                    break;
                case 5:
                    displayProducts();
                    break;
                case 6:
```

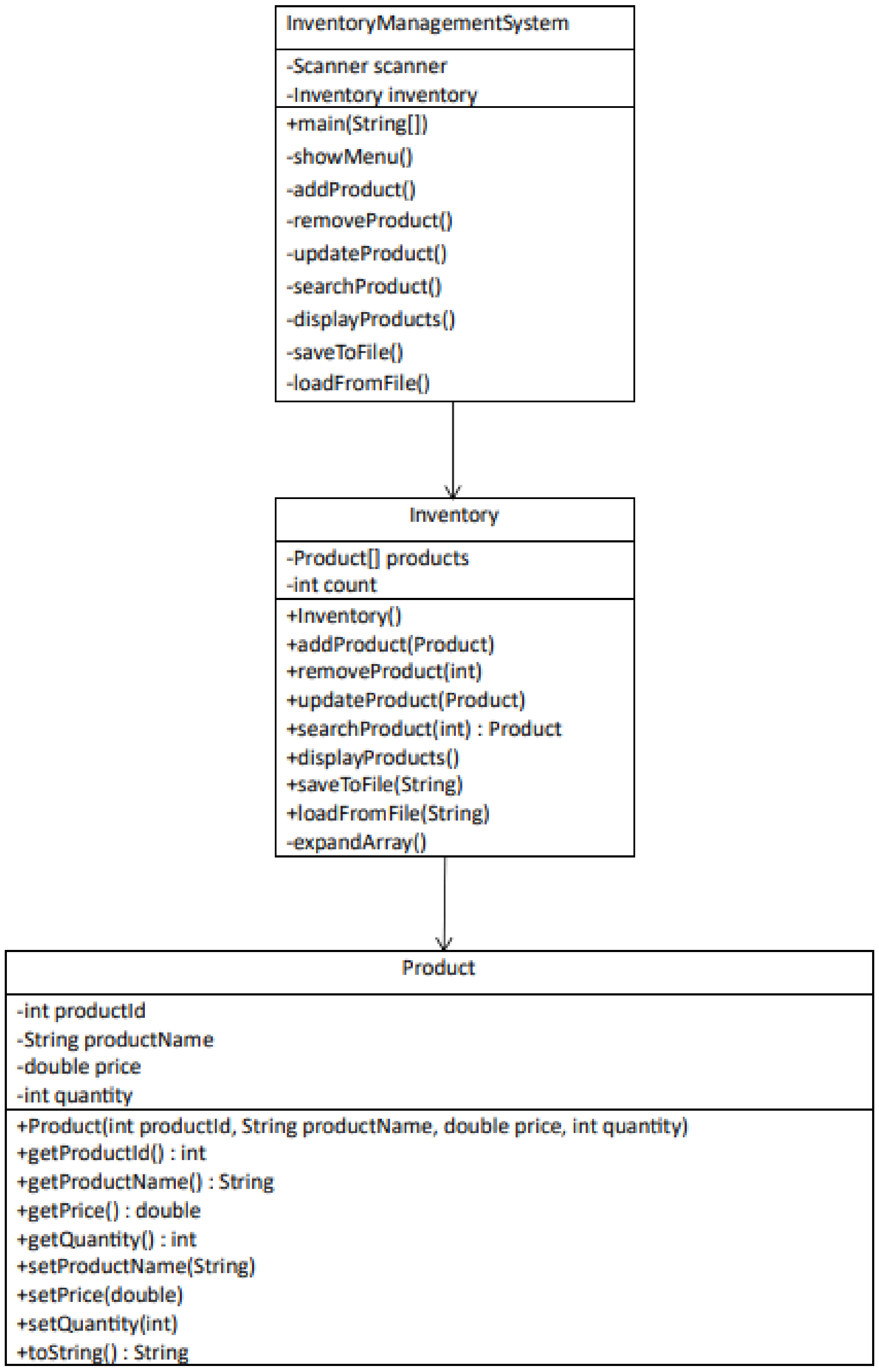
```
            saveToFile();
            break;
        case 7:
            loadFromFile();
            break;
        case 8:
            System.out.println("Exiting...");
            return;
        default:
            System.out.println("Invalid choice, please try again.");
      }
   }
}

private static void showMenu() {
   System.out.println("Inventory Management System");
   System.out.println("1. Add Product");
   System.out.println("2. Remove Product");
   System.out.println("3. Update Product");
   System.out.println("4. Search Product");
   System.out.println("5. Display Products");
   System.out.println("6. Save Products to File");
   System.out.println("7. Load Products from File");
   System.out.println("8. Exit");
   System.out.print("Enter your choice: ");
}

private static void addProduct() {
   System.out.print("Enter Product ID: ");
   int id = scanner.nextInt();
   scanner.nextLine();  // Consume newline
   System.out.print("Enter Product Name: ");
   String name = scanner.nextLine();
```

```
    System.out.print("Enter Product Price: ");
    double price = scanner.nextDouble();
    System.out.print("Enter Product Quantity: ");
    int quantity = scanner.nextInt();

    Product product = new Product(id, name, price, quantity);
    inventory.addProduct(product);
    System.out.println("Product added successfully.");
}

private static void removeProduct() {
    System.out.print("Enter Product ID to remove: ");
    int id = scanner.nextInt();
    inventory.removeProduct(id);
    System.out.println("Product removed successfully.");
}

private static void updateProduct() {
    System.out.print("Enter Product ID to update: ");
    int id = scanner.nextInt();
    scanner.nextLine();  // Consume newline
    System.out.print("Enter new Product Name: ");
    String name = scanner.nextLine();
    System.out.print("Enter new Product Price: ");
    double price = scanner.nextDouble();
    System.out.print("Enter new Product Quantity: ");
    int quantity = scanner.nextInt();

    Product updatedProduct = new Product(id, name, price, quantity);
    inventory.updateProduct(updatedProduct);
    System.out.println("Product updated successfully.");
}
```

```
private static void searchProduct() {
   System.out.print("Enter Product ID to search: ");
   int id = scanner.nextInt();
   Product product = inventory.searchProduct(id);
   if (product != null) {
      System.out.println("Product found: " + product);
   } else {
      System.out.println("Product not found.");
   }
}

private static void displayProducts() {
   inventory.displayProducts();
}

private static void saveToFile() {
   System.out.print("Enter filename to save: ");
   String filename = scanner.next();
   try {
      inventory.saveToFile(filename);
      System.out.println("Products saved successfully.");
   } catch (IOException e) {
      System.out.println("Error saving to file: " + e.getMessage());
   }
}

private static void loadFromFile() {
   System.out.print("Enter filename to load: ");
   String filename = scanner.next();
   try {
      inventory.loadFromFile(filename);
      System.out.println("Products loaded successfully.");
   } catch (IOException e) {
```

```
            System.out.println("Error loading from file: " + e.getMessage());
        }
    }
}
```

UML Diagram:


InventoryManagementSystem
-Scanner scanner
-Inventory inventory
+main(String[])
-showMenu()
-addProduct()
-removeProduct()
-updateProduct()
-searchProduct()
-displayProducts()
-saveToFile()
-loadFromFile()
Inventory
-Product[] products
-int count
+Inventory()
+addProduct(Product)
+removeProduct(int)
+updateProduct(Product)
+searchProduct(int) : Product
+displayProducts()
+saveToFile(String)
+loadFromFile(String)
-expandArray()
Product
-int productId
-String productName
-double price
-int quantity
+Product(int productId, String productName, double price, int quantity)
+getProductId() : int
+getProductName() : String
+getPrice() : double
+getQuantity() : int
+setProductName(String)
+setPrice(double)
+setQuantity(int)
+toString() : String