Project Report: Product Management System in Java

1) My Information

- **Name:** Raif El-khatib
- **Student ID:** GH1033530
- **Course:** Advanced programming
- **Submission Date:** 18/12/2025
- **GitHub Repository:** https://github.com/raifkhatib/java
- **Video Demonstration: https://youtu.be/pugDka4YUyY**

2) Introduction

For this project, I built a simple Product Management System using Java. The main goal was to let users add, list, search, sort products, and calculate their total value through a console menu that is so simple to use.

While working on it, I focused on using key Object Oriented Programming (OOP) ideas like classes, inheritance, encapsulation, and method overriding. I also practiced using loops, Array Lists, conditions, input validation, and lambda expressions to make the program work smoothly.

I decided to create a console-based application so I could concentrate on the logic and data handling instead of designing a graphical interface. Even though it is not like a hard professional advanced project, it can still show important Java skills, and the code is organized, clear, and easy to update.

3. System Architecture

The system was built using a modular approach, where each part of the program handles its own specific task. This makes the code easier to understand, maintain, and improve in the future. The main parts of the system include:

Item Class (Base Class)

- Holds the basic details of an item, such as its ID and name.
- Includes a displayinfo() method that prints the item's information.
- This class shows core OOP ideas like encapsulation and simple class design.

Product Class (Subclass)

- Inherits from the `Item` class and adds a price attribute.
- Overrides displayinfo() to also show the product's price.
- Demonstrates inheritance, the use of Super() in constructors, and method overriding.

Main Program / User Interface

- Uses a menu-based console interface controlled by a `while` loop.
- Offers features such as adding products, listing all products, searching by ID, sorting by price, showing the total value, and exiting the program.
- Stores products in an ArrayList<Product> for flexible and dynamic data handling.
- Includes input validation to prevent errors such as duplicate IDs, invalid numbers, or empty names.

The architecture can be represented as:

[User Input] → [Menu System] → [Methods: add, list, search, sort, total] → [ArrayList storage] → [Console Output]

4. Implementation

Now, I will explain how I implemented the project in short:

1) Base Class Item

The first step was to create a base class named Item. This class represents a generic item with two main attributes that are ID and Name.

```
static class Item {
  protected String name;
  protected int id;

  public Item(String name, int id) {
    this.name = name;
    this.id = id;
  }

  public void displayInfo() {
    System.out.println("ID: " + id + " | Name: " + name);
  }
}
```

Explanation:

- I used protected variables so that subclasses can easily access them when needed.
- The constructor sets up the item by assigning its ID and name.
- The displayinfo() method prints the item's ID and name to the console.
- This part of the project shows the basics of class creation, using constructors, working with access modifiers, and defining methods in Java.

2) Subclass Product

Next, I created a Product class that extends the Item class. This adds a price field to each product, demonstrating inheritance.

```
static class Product extends Item {
  private final double price;


  public Product(String name, int id, double price) {
    super(name, id);
    this.price = price;
  }


  @Override
  public void displayInfo() {
    super.displayInfo();
    System.out.println("Price: $" + price);
  }


  public double getPrice() { return price; }
}
```

Explanation:

- (extends item) shows that the Product class inherits from the Item class.
- Super(name,id) is used to call the parent class constructor so the inherited fields are properly initialized.
- @override displayinfo() updates the method so it also shows the product price when printing the information.
- The price variable is private and accessed through a getter method, which demonstrates encapsulation.

This highlights that a product is simply a more specialized type of item.

3) Menu and Product Management

The main part of the program is a menu-based system that lets users interact with the products easily.

```
static Scanner scanner = new Scanner(System.in);

static ArrayList<Product> products = new ArrayList<>();
```

The program has a simple menu to let users interact with the products. You can add a new product, list all products, search for a product by ID, exit the program, see products sorted by price, or view the total value of all products. The menu keeps running until the user chooses to exit. Depending on the user choice, the program runs the right action. Each task is handled by its own method, which makes the code easy to follow and keeps everything organized.

4) Input Validation and Error Handling

To make sure the program runs smoothly, I added input validation when users add new products.

```
while (true) {
  System.out.print("Enter product ID: ");
  if (scanner.hasNextInt()) {
    id = scanner.nextInt();
    scanner.nextLine();
    // Check for duplicate ID
    boolean exists = false;
    for (Product p : products) {
      if (p.id == id) { exists = true; break; }
    }
    if (!exists) break;
    System.out.println("That ID is already taken. Try again.");
  } else {
    System.out.println("Invalid input! Enter a number.");
    scanner.nextLine();
  }
}
```

Explanation:

- Checks that the product ID is a valid number.
- Prevents duplicate IDs by using a Boolean flag.
- Ensures the product name is not empty and the price is a number greater than or equal to zero.
- Uses loops, conditional statements, and Scanner methods to handle input safely.

5) Optional Features Sorting and Total Value

I added extra features to make the program more complete and user-friendly.

Sorting products by price:

```
products.sort((p1, p2) -> Double.compare(p1.getPrice(), p2.getPrice()));
```

Calculating total value:

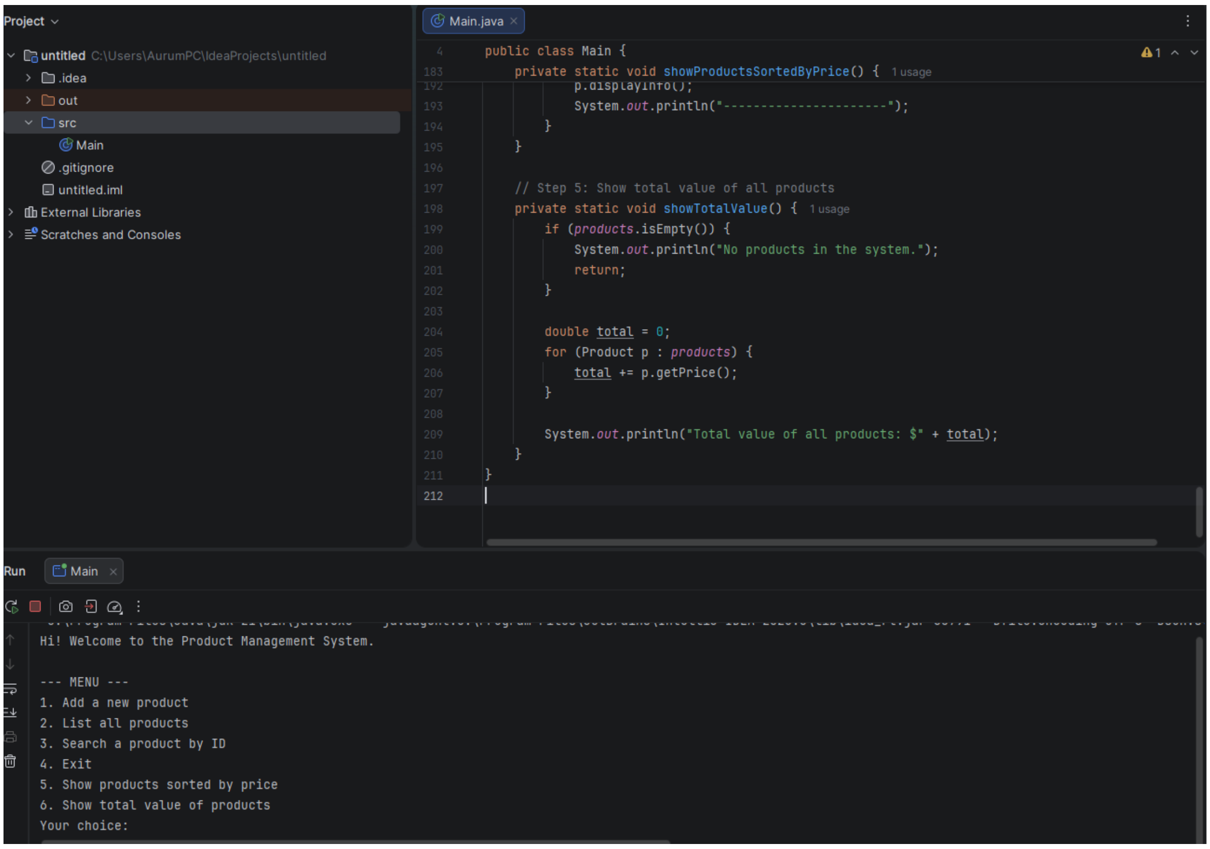
```
double total = 0;
for (Product p : products) {
  total += p.getPrice();
}
System.out.println("Total value: $" + total);
```

Explanation:

- Sorting is done using a lambda expression with Arraylist.sort().
- The total value is calculated with a for-each loop that sums up all product prices.
- These features demonstrate understanding of Java collections, loops, and lambda expressions.

5) Results

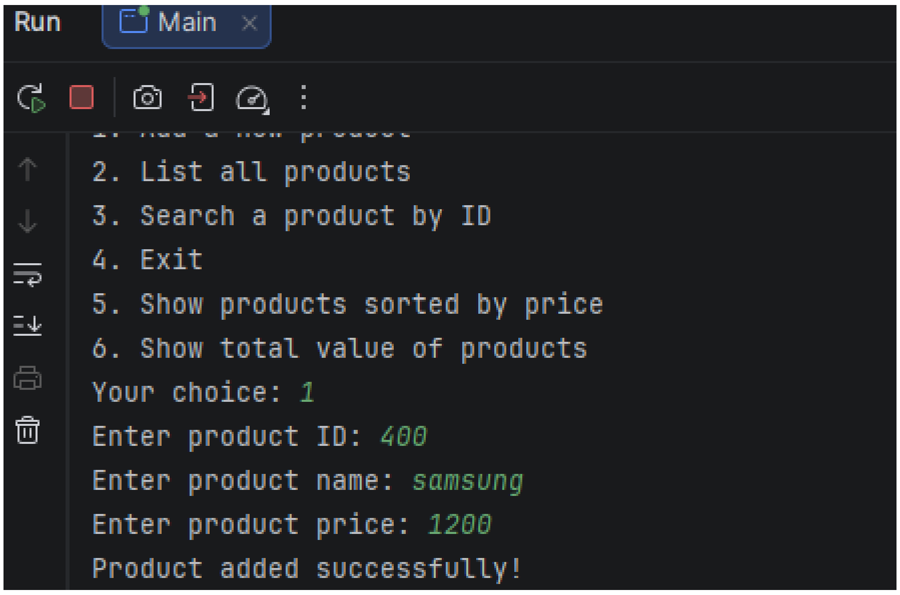
Here are the screenshots that show the program running.



```java
public class Main {
    private static void showProductsSortedByPrice() {
                p.displayInfo();
                System.out.println("---------------------");
            }
        }

        // Step 5: Show total value of all products
        private static void showTotalValue() {
            if (products.isEmpty()) {
                System.out.println("No products in the system.");
                return;
            }

            double total = 0;
            for (Product p : products) {
                total += p.getPrice();
            }

            System.out.println("Total value of all products: $" + total);
        }
}
```

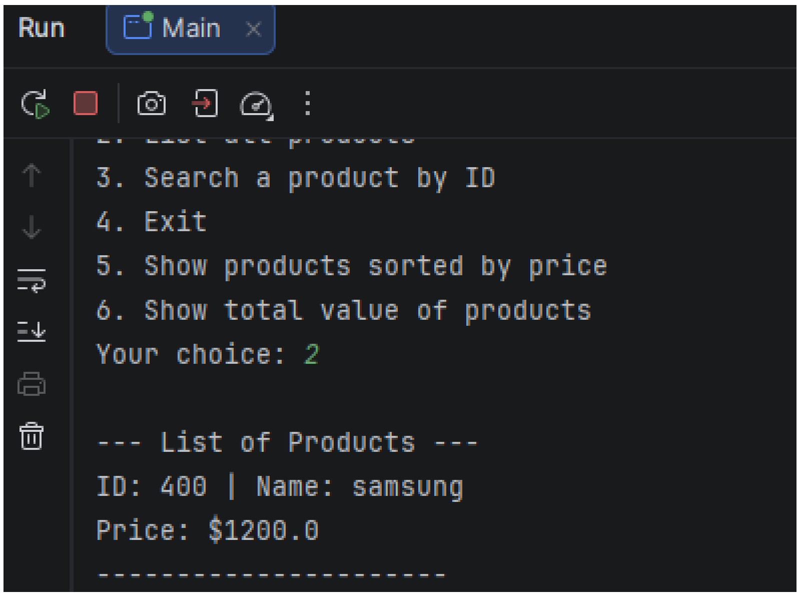
```
Hi! Welcome to the Product Management System.

--- MENU ---
1. Add a new product
2. List all products
3. Search a product by ID
4. Exit
5. Show products sorted by price
6. Show total value of products
Your choice:
```

This shows that the code runs perfectly.



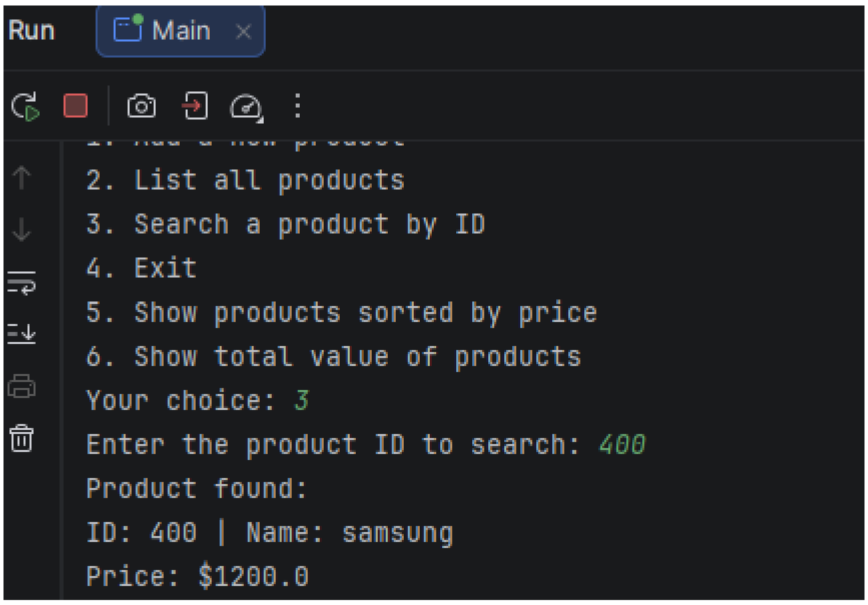
```
2. List all products
3. Search a product by ID
4. Exit
5. Show products sorted by price
6. Show total value of products
Your choice: 1
Enter product ID: 400
Enter product name: samsung
Enter product price: 1200
Product added successfully!
```

Here I did the 1st choice which is adding a new product with ID, name, and price.



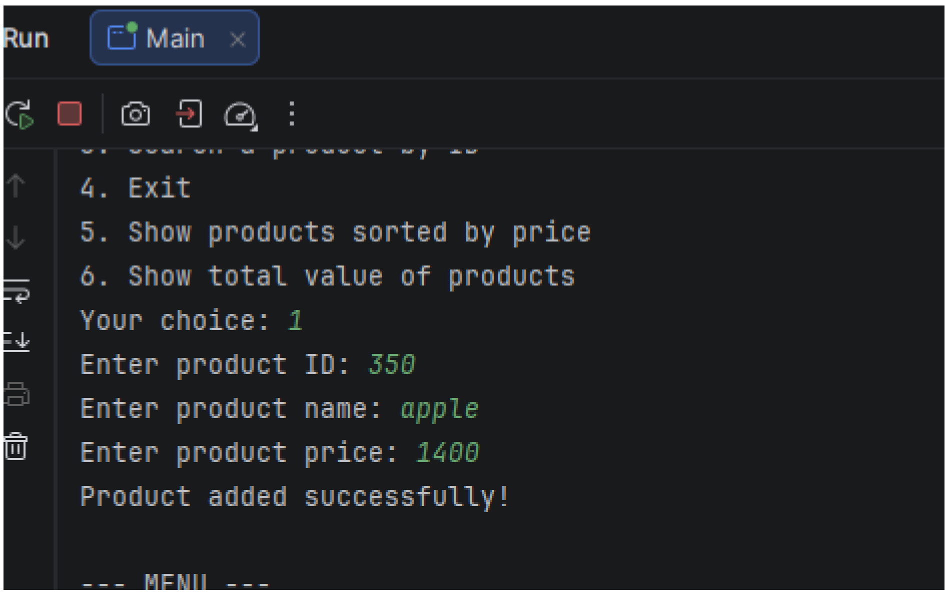
```
3. Search a product by ID
4. Exit
5. Show products sorted by price
6. Show total value of products
Your choice: 2

--- List of Products ---
ID: 400 | Name: samsung
Price: $1200.0
----------------------
```

And here I listed all products in the system.



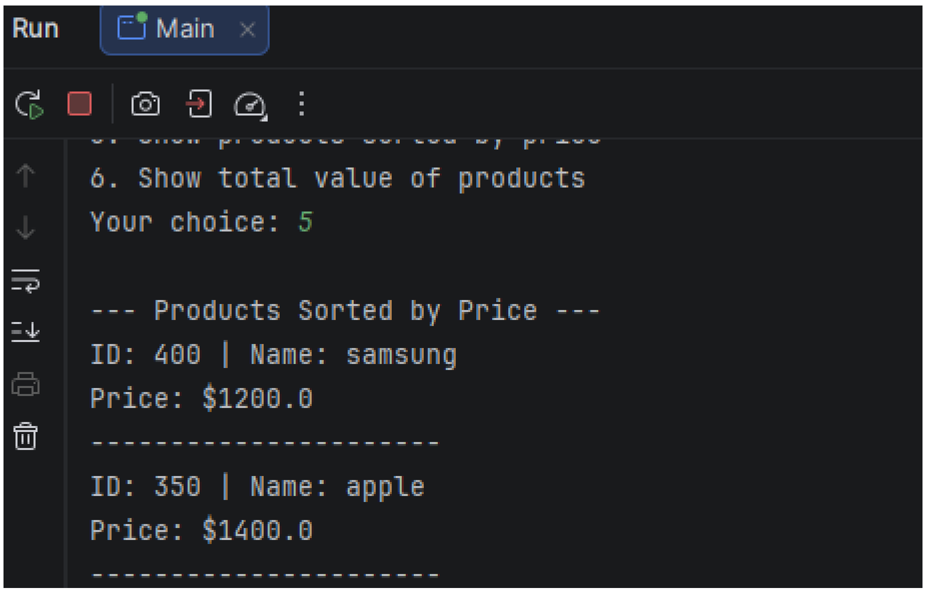
```
2. List all products
3. Search a product by ID
4. Exit
5. Show products sorted by price
6. Show total value of products
Your choice: 3
Enter the product ID to search: 400
Product found:
ID: 400 | Name: samsung
Price: $1200.0
```

Here I searched for a product by ID.



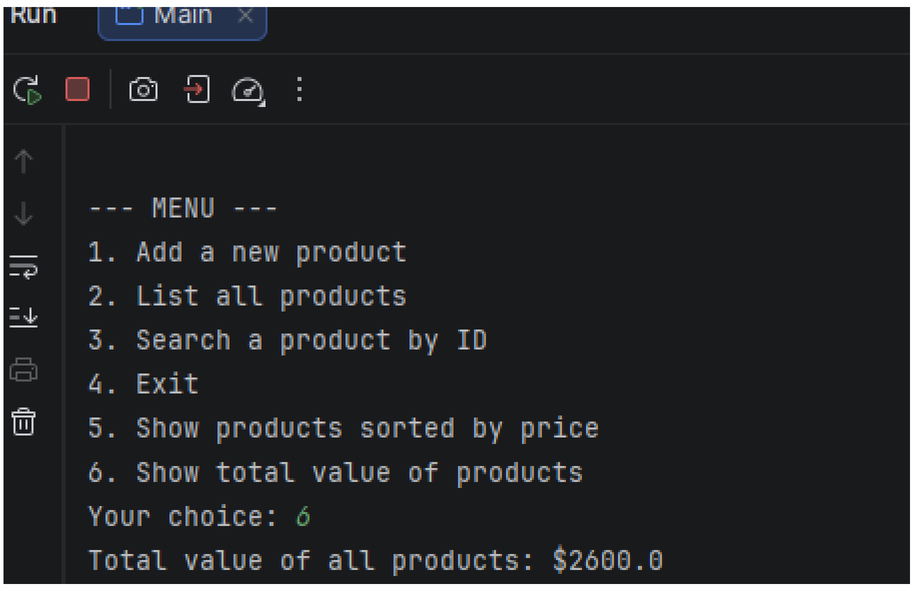
```
4. Exit
5. Show products sorted by price
6. Show total value of products
Your choice: 1
Enter product ID: 350
Enter product name: apple
Enter product price: 1400
Product added successfully!

--- MENU ---
```

I added another product so i can sort products next.



```
6. Show total value of products
Your choice: 5

--- Products Sorted by Price ---
ID: 400 | Name: samsung
Price: $1200.0
----------------------
ID: 350 | Name: apple
Price: $1400.0
----------------------
```

And here I sorted the products by price.



```
--- MENU ---
1. Add a new product
2. List all products
3. Search a product by ID
4. Exit
5. Show products sorted by price
6. Show total value of products
Your choice: 6
Total value of all products: $2600.0
```

Finally, the Total value of all products.

6) Challenges and solutions

While building the Product Management System, I faced a few challenges. One was making sure the program handled input correctly, like avoiding duplicate IDs, checking for valid numbers, and making sure product names were not empty. I solved this by adding loops with clear messages to guide the user. Another challenge was managing the Scanner properly so numbers and text would not mix up. Sorting products by price and calculating the total value also needed careful planning of course, which I solved by using an ArrayList and Java's sorting methods.

7) Conclusion and Future Work

In this project, I successfully built a Product Management System that allows users to add, list, search, and manage products with proper input validation. I also implemented features to sort products by price and calculate the total value, making the system more useful and organized.

I know this is not a big or an advanced level project, it is just that i am a beginner in this, and going for an easy task was a good way to start. Through this project, I learned how to use concepts like classes, inheritance, and method overriding, as well as how to handle user input and manage lists effectively in Java.

For future work, I could change or add some stuff like removing or updating products, searching by name, saving data to a file, or adding a friendly graphical interface to make it even more interactive.