**Exercise 1: Inventory Management System**

* **CODE FOR Product.java**

public class Product {

    private int productId;

    private String productName;

    private int quantity;

    private double price;

    public Product(int productId, String productName, int quantity, double price) {

        this.productId = productId;

        this.productName = productName;

        this.quantity = quantity;

        this.price = price;

    }

    public int getProductId() { return productId; }

    public String getProductName() { return productName; }

    public int getQuantity() { return quantity; }

    public double getPrice() { return price; }

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

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

    @Override

    public String toString() {

        return "Product ID: " + productId +

            ", Name: " + productName +

            ", Quantity: " + quantity +

            ", Price: Rs." + price;

    }

}

* **CODE FOR Inventory.java**

import java.util.HashMap;

public class Inventory {

    private HashMap<Integer, Product> inventory = new HashMap<>();

    // Add product

    public void addProduct(Product product) {

        inventory.put(product.getProductId(), product);

        System.out.println("Added: " + product.getProductName());

    }

    // Update product quantity and price

    public void updateProduct(int productId, int newQty, double newPrice) {

        if (inventory.containsKey(productId)) {

            Product p = inventory.get(productId);

            p.setQuantity(newQty);

            p.setPrice(newPrice);

            System.out.println("Updated: " + p);

        } else {

            System.out.println("Product not found.");

        }

    }

    // Delete product

    public void deleteProduct(int productId) {

        if (inventory.remove(productId) != null) {

            System.out.println("Deleted product ID: " + productId);

        } else {

            System.out.println("Product not found.");

        }

    }

    // View all products

    public void displayInventory() {

        if (inventory.isEmpty()) {

            System.out.println("Inventory is empty.");

        } else {

            System.out.println("Current Inventory:");

            for (Product p : inventory.values()) {

                System.out.println(p);

            }

        }

    }

}

* **CODE FOR Main.java**

public class Main {

    public static void main(String[] args) {

        Inventory manager = new Inventory();

        Product p1 = new Product(01, "Book", 10, 500);

        Product p2 = new Product(02, "Pen", 50, 10);

        Product p3 = new Product(03, "Laptop", 30, 72000);

        manager.addProduct(p1);

        manager.addProduct(p2);

        manager.addProduct(p3);

        manager.displayInventory();

        manager.updateProduct(02, 60, 20);

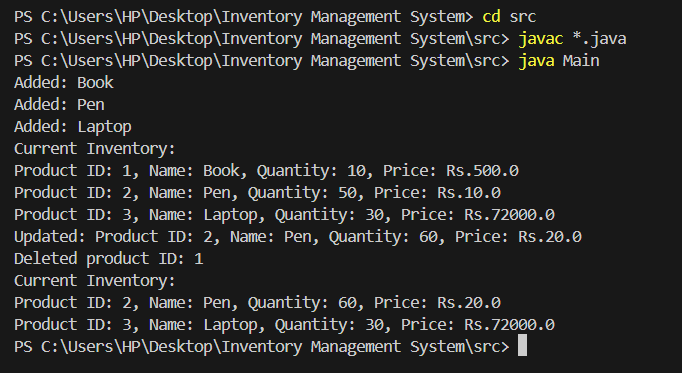
        manager.deleteProduct(01);

        manager.displayInventory();

    }

}

* **OUTPUT**



**Exercise 2: E-commerce Platform Search Function**

* **CODE FOR Product.java**

public class Product {

    int productId;

    String productName;

    String category;

    public Product(int productId, String productName, String category) {

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    @Override

    public String toString() {

        return "Product ID: " + productId + ", Name: " + productName + ", Category: " + category;

    }

}

* **CODE FOR ProductSearch.java**

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

     // Linear Search by Product Name

    public static Product linearSearch(Product[] products, String targetName) {

        for (Product p : products) {

            if (p.productName.equalsIgnoreCase(targetName)) {

                return p;

            }

        }

        return null;

    }

    // Binary Search by Product Name

    public static Product binarySearch(Product[] products, String targetName) {

        int left = 0;

        int right = products.length - 1;

        while (left <= right) {

            int mid = (left + right) / 2;

            int cmp = products[mid].productName.compareToIgnoreCase(targetName);

            if (cmp == 0) return products[mid];

            else if (cmp < 0) left = mid + 1;

            else right = mid - 1;

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] products = {

            new Product(101, "Laptop", "Electronics"),

            new Product(102, "Shampoo", "Personal Care"),

            new Product(103, "Notebook", "Stationery"),

            new Product(104, "Smartphone", "Electronics"),

            new Product(105, "Tablet", "Electronics")

        };

        // Sort for binary search

        Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

        // Perform Linear Search

        Product result1 = linearSearch(products, "Notebook");

        System.out.println("Linear Search Result: " + (result1 != null ? result1 : "Not Found"));

        // Perform Binary Search

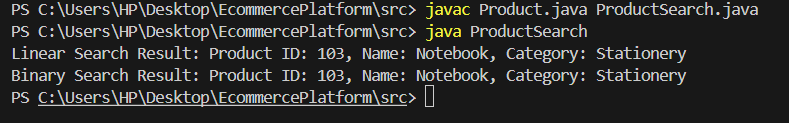
        Product result2 = binarySearch(products, "Notebook");

        System.out.println("Binary Search Result: " + (result2 != null ? result2 : "Not Found"));

    }

}

* **OUTPUT**

****

**Exercise 7: Financial Forecasting**

* CODE FOR Financial Forecasting

public class FinancialForecast {

    public static double futureValue(double presentValue, double growthRate, int years) {

        if (years == 0) {

            return presentValue;

        } else {

            return futureValue(presentValue \* (1 + growthRate), growthRate, years - 1);

        }

    }

    public static void main(String[] args) {

        double presentValue = 10000;

        double growthRate = 0.07;

        int years = 5;

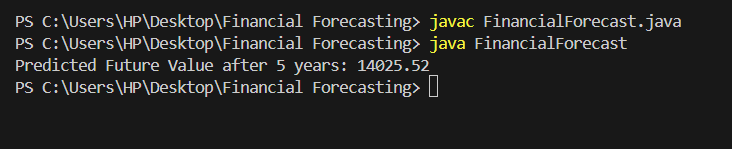
        double predictedValue = futureValue(presentValue, growthRate, years);

        System.out.printf("Predicted Future Value after %d years: %.2f\n", years, predictedValue);

    }

}

**OUTPUT**

****