**Exercise 1: Inventory Management System**

**InventoryManagementSystem.java**

import java.util.HashMap;

import java.util.Map;

class Product {

int productId;

String productName;

int quantity;

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 void display() {

System.out.println("ID: " + productId + ", Name: " + productName +

", Quantity: " + quantity + ", Price: ₹" + price);

}

}

public class InventoryManagementSystem {

private Map<Integer, Product> inventory;

public InventoryManagementSystem() {

inventory = new HashMap<>();

}

// Add a product

public void addProduct(Product product) {

if (inventory.containsKey(product.productId)) {

System.out.println("Product ID already exists. Use updateProduct instead.");

} else {

inventory.put(product.productId, product);

System.out.println("Product added successfully.");

}

}

// Update a product

public void updateProduct(int productId, int quantity, double price) {

Product product = inventory.get(productId);

if (product != null) {

product.quantity = quantity;

product.price = price;

System.out.println("Product updated successfully.");

} else {

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

}

}

// Delete a product

public void deleteProduct(int productId) {

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

System.out.println("Product deleted successfully.");

} else {

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

}

}

// Display all products

public void displayInventory() {

if (inventory.isEmpty()) {

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

} else {

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

product.display();

}

}

}

// Main method for testing

public static void main(String[] args) {

InventoryManagementSystem ims = new InventoryManagementSystem();

Product p1 = new Product(101, "Mouse", 50, 299.99);

Product p2 = new Product(102, "Keyboard", 30, 499.49);

ims.addProduct(p1);

ims.addProduct(p2);

ims.displayInventory();

ims.updateProduct(101, 60, 279.99);

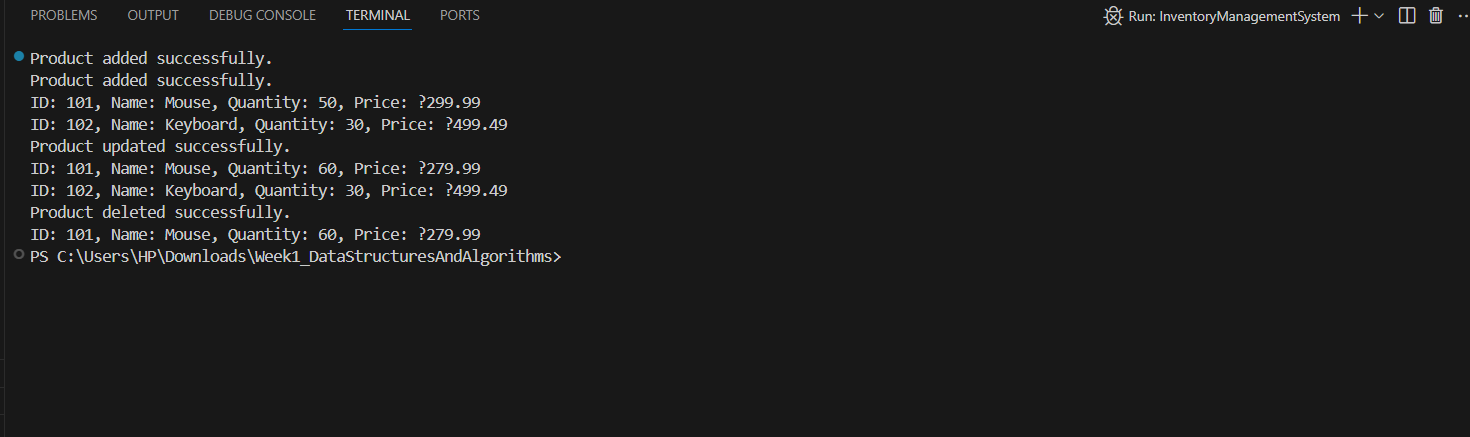
ims.displayInventory();

ims.deleteProduct(102);

ims.displayInventory();

}

}  
**Output:**



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

**Product.java**

package EcommerceSearchFunction;

public class Product {

public int productId;

public String productName;

public 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;

}

}

**SearchUtility.java**

package EcommerceSearchFunction;

public class SearchUtility {

// Linear search

public static Product linearSearch(Product[] products, int targetId) {

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

// Binary search (requires sorted array)

public static Product binarySearch(Product[] products, int targetId) {

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (products[mid].productId == targetId) return products[mid];

if (products[mid].productId < targetId) left = mid + 1;

else right = mid - 1;

}

return null;

}

}

**TestSearch.java**

package EcommerceSearchFunction;

import java.util.Arrays;

public class TestSearch {

public static void main(String[] args) {

Product[] products = {

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

new Product(203, "Shoes", "Footwear"),

new Product(145, "Smartphone", "Electronics"),

new Product(312, "T-shirt", "Clothing"),

};

// Sort by productId for binary search

Arrays.sort(products, (a, b) -> Integer.compare(a.productId, b.productId));

// Linear Search

Product result1 = SearchUtility.linearSearch(products, 145);

System.out.println("Linear Search Result: " + (result1 != null ? result1 : "Product not found"));

// Binary Search

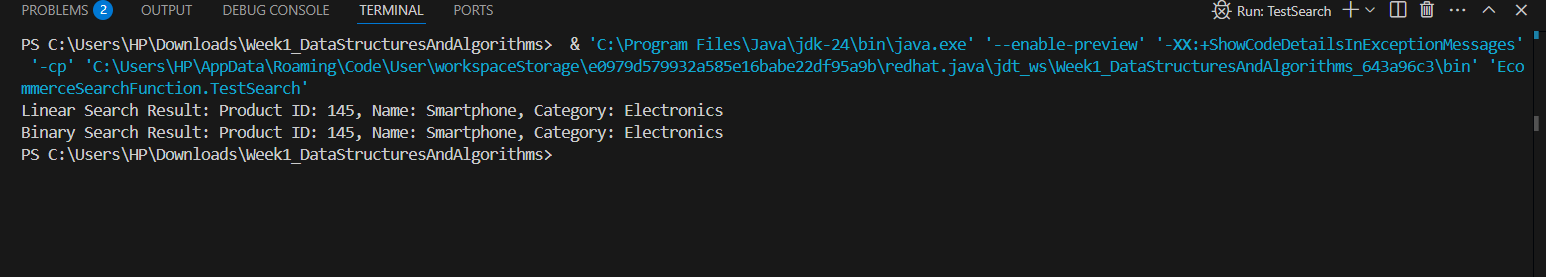
Product result2 = SearchUtility.binarySearch(products, 145);

System.out.println("Binary Search Result: " + (result2 != null ? result2 : "Product not found"));

}

}

**Output:**

****

**Exercise 3: Sorting Customer Orders**

**OrderSortingSystem.java**

class Order {

int orderId;

String customerName;

double totalPrice;

public Order(int orderId, String customerName, double totalPrice) {

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public void display() {

System.out.println("OrderID: " + orderId + ", Customer: " + customerName + ", Price: ₹" + totalPrice);

}

}

public class OrderSortingSystem {

// Bubble Sort

public static void bubbleSort(Order[] orders) {

int n = orders.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (orders[j].totalPrice > orders[j + 1].totalPrice) {

Order temp = orders[j];

orders[j] = orders[j + 1];

orders[j + 1] = temp;

}

}

}

}

// Quick Sort

public static void quickSort(Order[] orders, int low, int high) {

if (low < high) {

int pi = partition(orders, low, high);

quickSort(orders, low, pi - 1);

quickSort(orders, pi + 1, high);

}

}

private static int partition(Order[] orders, int low, int high) {

double pivot = orders[high].totalPrice;

int i = low - 1;

for (int j = low; j < high; j++) {

if (orders[j].totalPrice <= pivot) {

i++;

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

Order temp = orders[i + 1];

orders[i + 1] = orders[high];

orders[high] = temp;

return i + 1;

}

public static void main(String[] args) {

Order[] orders = {

new Order(1, "Alice", 299.99),

new Order(2, "Bob", 499.49),

new Order(3, "Charlie", 199.99),

new Order(4, "David", 799.00)

};

System.out.println("Before Sorting:");

for (Order o : orders) o.display();

// You can test either sort:

// bubbleSort(orders);

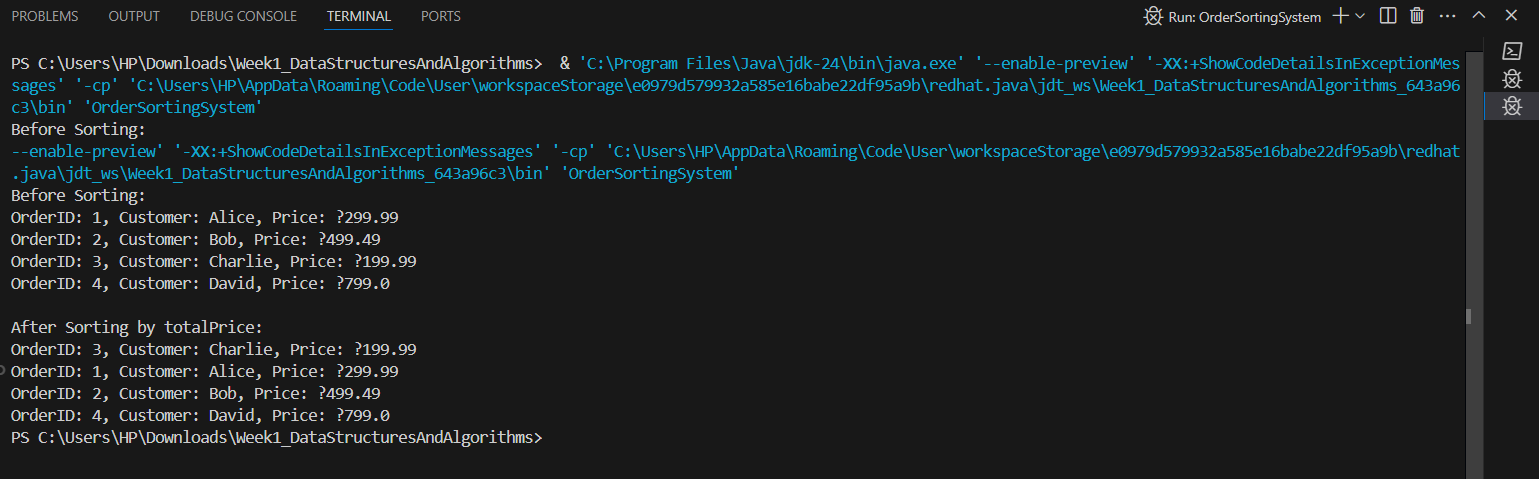
quickSort(orders, 0, orders.length - 1);

System.out.println("\nAfter Sorting by totalPrice:");

for (Order o : orders) o.display();

}

}  
**Output:**

****

**Exercise 4: Employee Management System**

**EmployeeManagementSystem.java**

class Employee {

int employeeId;

String name;

String position;

double salary;

public Employee(int employeeId, String name, String position, double salary) {

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

public void display() {

System.out.println("ID: " + employeeId + ", Name: " + name +

", Position: " + position + ", Salary: ₹" + salary);

}

}

public class EmployeeManagementSystem {

private Employee[] employees;

private int count;

public EmployeeManagementSystem(int size) {

employees = new Employee[size];

count = 0;

}

// Add

public void addEmployee(Employee emp) {

if (count < employees.length) {

employees[count++] = emp;

System.out.println("Employee added.");

} else {

System.out.println("Employee list is full.");

}

}

// Search

public void searchEmployee(int id) {

for (int i = 0; i < count; i++) {

if (employees[i].employeeId == id) {

employees[i].display();

return;

}

}

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

}

// Display all

public void displayAll() {

for (int i = 0; i < count; i++) {

employees[i].display();

}

}

// Delete

public void deleteEmployee(int id) {

for (int i = 0; i < count; i++) {

if (employees[i].employeeId == id) {

for (int j = i; j < count - 1; j++) {

employees[j] = employees[j + 1];

}

employees[--count] = null;

System.out.println("Employee deleted.");

return;

}

}

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

}

public static void main(String[] args) {

EmployeeManagementSystem ems = new EmployeeManagementSystem(5);

ems.addEmployee(new Employee(1, "John", "Manager", 60000));

ems.addEmployee(new Employee(2, "Jane", "Developer", 50000));

ems.addEmployee(new Employee(3, "Mike", "Tester", 40000));

System.out.println("\nAll Employees:");

ems.displayAll();

System.out.println("\nSearch Employee ID 2:");

ems.searchEmployee(2);

System.out.println("\nDelete Employee ID 1:");

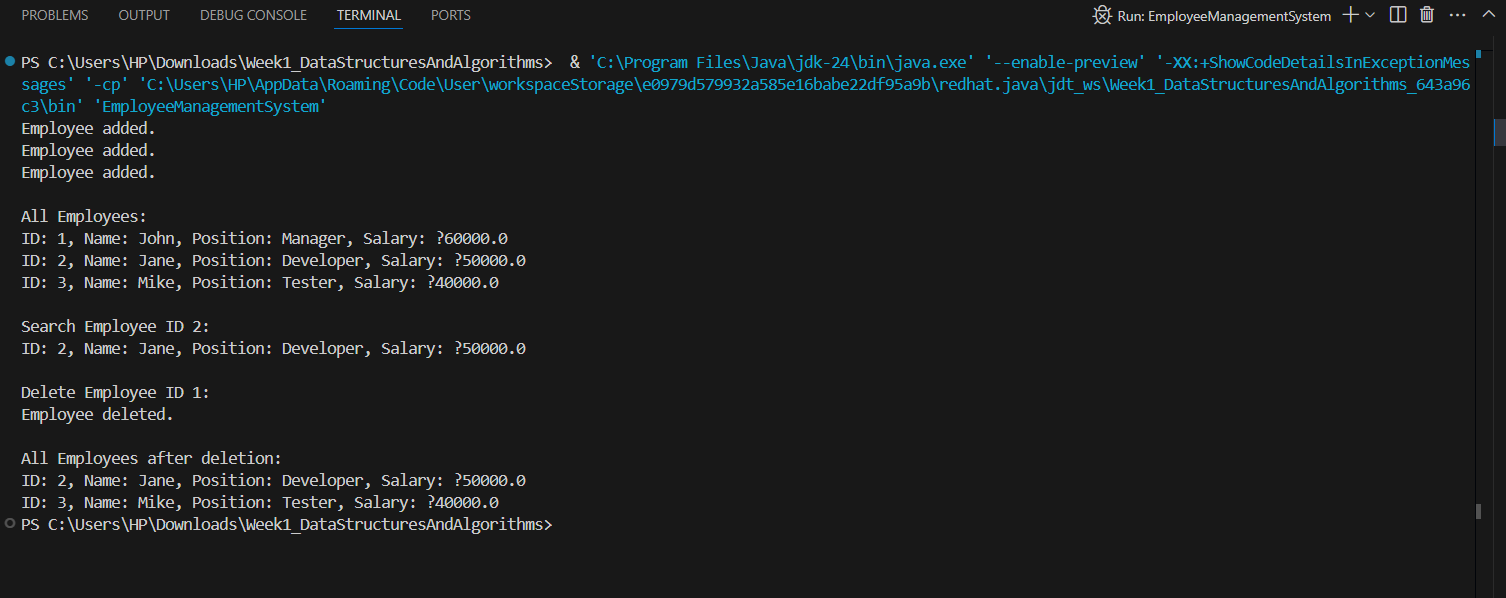
ems.deleteEmployee(1);

System.out.println("\nAll Employees after deletion:");

ems.displayAll();

}

}  
**Output:**

****

**Exercise 5: Task Management System  
TaskManagementSystem.java**

class Task {

int taskId;

String taskName;

String status;

Task next;

public Task(int taskId, String taskName, String status) {

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

this.next = null;

}

public void display() {

System.out.println("ID: " + taskId + ", Name: " + taskName + ", Status: " + status);

}

}

public class TaskManagementSystem {

private Task head;

// Add task at end

public void addTask(int id, String name, String status) {

Task newTask = new Task(id, name, status);

if (head == null) {

head = newTask;

} else {

Task temp = head;

while (temp.next != null) temp = temp.next;

temp.next = newTask;

}

System.out.println("Task added.");

}

// Search by ID

public void searchTask(int id) {

Task temp = head;

while (temp != null) {

if (temp.taskId == id) {

temp.display();

return;

}

temp = temp.next;

}

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

}

// Display all tasks

public void displayTasks() {

if (head == null) {

System.out.println("No tasks found.");

return;

}

Task temp = head;

while (temp != null) {

temp.display();

temp = temp.next;

}

}

// Delete task by ID

public void deleteTask(int id) {

if (head == null) {

System.out.println("No tasks to delete.");

return;

}

if (head.taskId == id) {

head = head.next;

System.out.println("Task deleted.");

return;

}

Task current = head;

while (current.next != null && current.next.taskId != id) {

current = current.next;

}

if (current.next == null) {

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

} else {

current.next = current.next.next;

System.out.println("Task deleted.");

}

}

public static void main(String[] args) {

TaskManagementSystem tms = new TaskManagementSystem();

tms.addTask(101, "Design UI", "Pending");

tms.addTask(102, "Write backend", "In Progress");

tms.addTask(103, "Testing", "Pending");

System.out.println("\nAll Tasks:");

tms.displayTasks();

System.out.println("\nSearch Task ID 102:");

tms.searchTask(102);

System.out.println("\nDelete Task ID 101:");

tms.deleteTask(101);

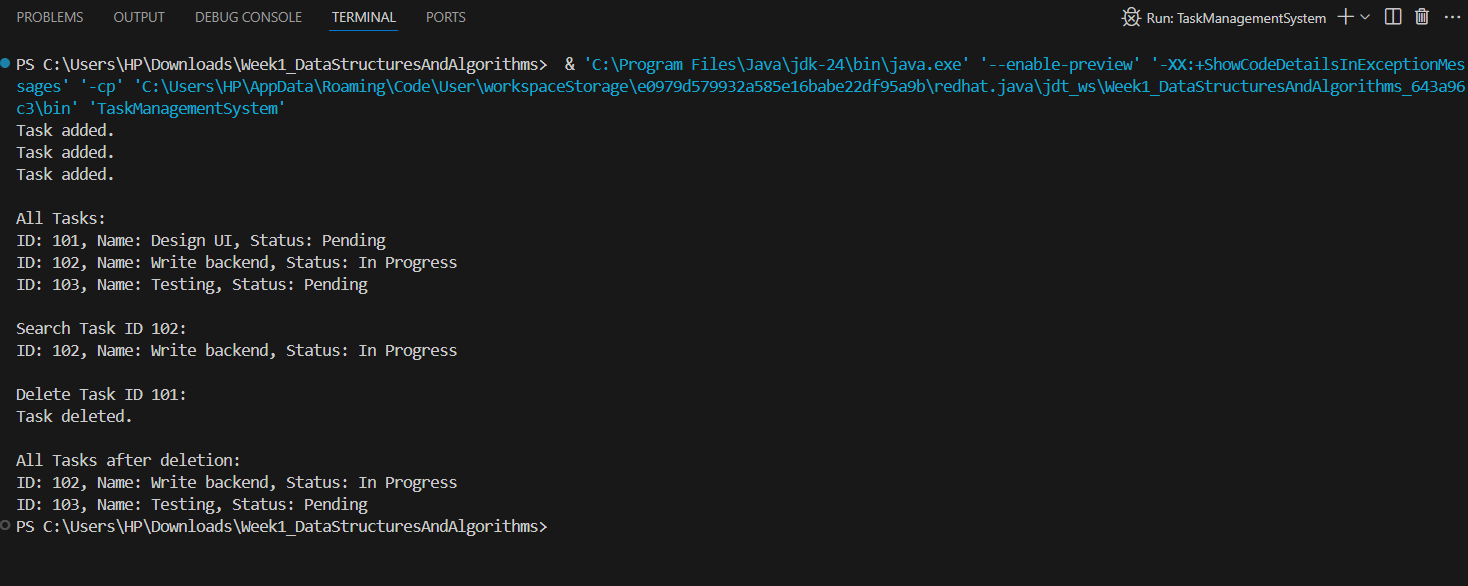
System.out.println("\nAll Tasks after deletion:");

tms.displayTasks();

}

}

**Output:**

****

**Exercise 6: Library Management System**

**LibraryManagementSystem.java**

import java.util.Arrays;

import java.util.Comparator;

class Book {

int bookId;

String title;

String author;

public Book(int bookId, String title, String author) {

this.bookId = bookId;

this.title = title;

this.author = author;

}

public void display() {

System.out.println("ID: " + bookId + ", Title: " + title + ", Author: " + author);

}

}

public class LibraryManagementSystem {

Book[] books;

int count;

public LibraryManagementSystem(int size) {

books = new Book[size];

count = 0;

}

// Add book

public void addBook(Book b) {

if (count < books.length) {

books[count++] = b;

} else {

System.out.println("Library is full.");

}

}

// Linear Search by Title

public void linearSearch(String title) {

for (int i = 0; i < count; i++) {

if (books[i].title.equalsIgnoreCase(title)) {

books[i].display();

return;

}

}

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

}

// Binary Search (assumes sorted by title)

public void binarySearch(String title) {

Arrays.sort(books, 0, count, Comparator.comparing(b -> b.title.toLowerCase()));

int left = 0, right = count - 1;

while (left <= right) {

int mid = (left + right) / 2;

int cmp = title.compareToIgnoreCase(books[mid].title);

if (cmp == 0) {

books[mid].display();

return;

} else if (cmp < 0) {

right = mid - 1;

} else {

left = mid + 1;

}

}

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

}

public static void main(String[] args) {

LibraryManagementSystem lms = new LibraryManagementSystem(5);

lms.addBook(new Book(1, "Java Basics", "John"));

lms.addBook(new Book(2, "Data Structures", "Alice"));

lms.addBook(new Book(3, "Python Intro", "Bob"));

System.out.println("Linear Search for 'Data Structures':");

lms.linearSearch("Data Structures");

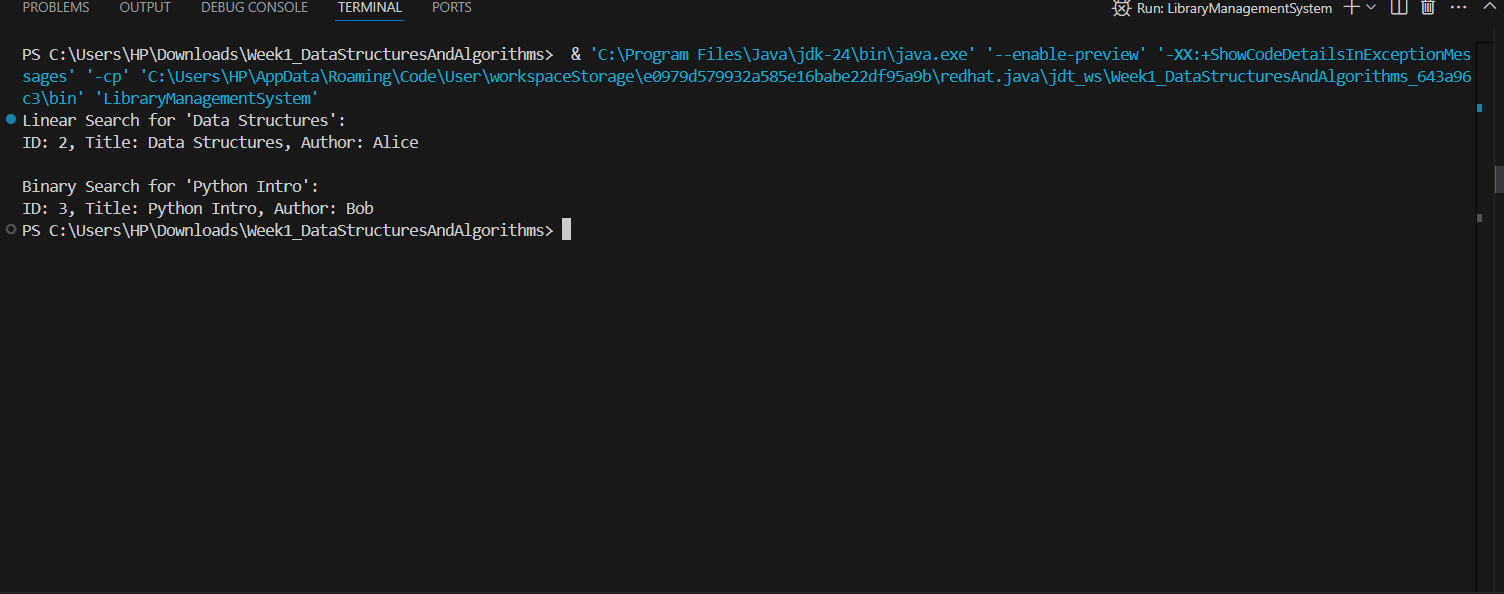
System.out.println("\nBinary Search for 'Python Intro':");

lms.binarySearch("Python Intro");

}

}

**Output**

****

**Exercise 7: Financial Forecasting**

**ForecastUtility.java**

package FinancialForecasting;

public class ForecastUtility {

// Recursive forecast: FV = PV \* (1 + r)^n

public static double forecastRecursive(double pv, double rate, int years) {

if (years == 0) return pv;

return forecastRecursive(pv, rate, years - 1) \* (1 + rate);

}

// Optimized iterative approach

public static double forecastIterative(double pv, double rate, int years) {

for (int i = 0; i < years; i++) {

pv \*= (1 + rate);

}

return pv;

}

}

**TestForecast.java**

package FinancialForecasting;

public class TestForecast {

public static void main(String[] args) {

double pv = 10000; // Present value

double rate = 0.05; // Annual interest rate (5%)

int years = 5;

double recursiveResult = ForecastUtility.forecastRecursive(pv, rate, years);

double iterativeResult = ForecastUtility.forecastIterative(pv, rate, years);

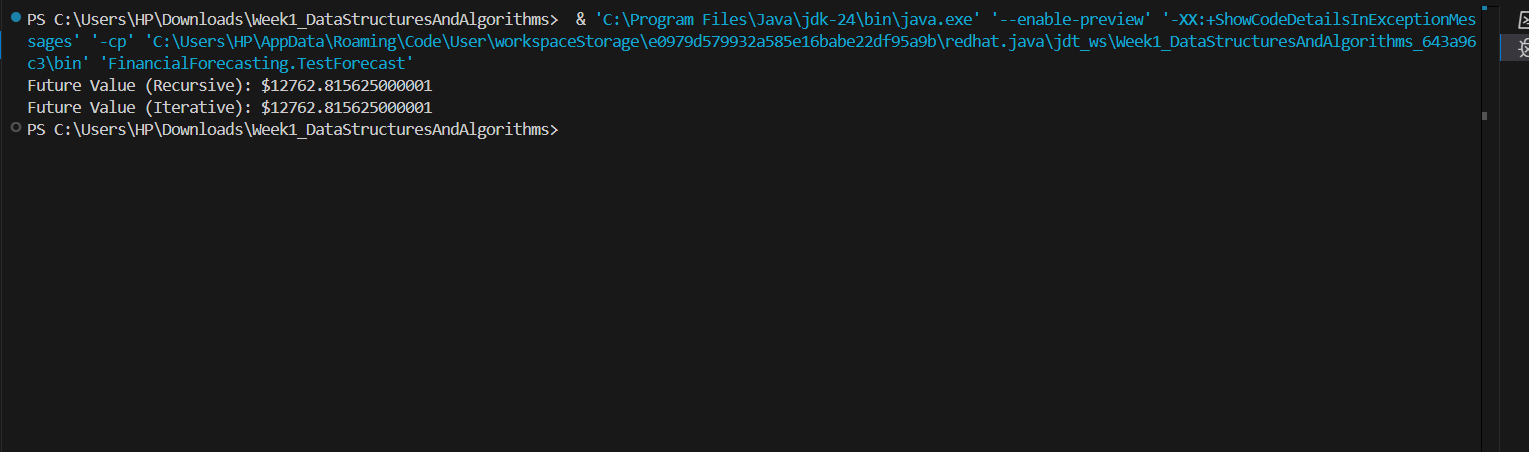
System.out.println("Future Value (Recursive): $" + recursiveResult);

System.out.println("Future Value (Iterative): $" + iterativeResult);

}

}

**Output:**

****