**Data Structures and Algorithm solution**

**-Harini Baskar(6396726)**

**Exercise 1: Inventory Management System**

**Code:**

Package Inventory;

import java.util.\*;

public class Product {

int productId;

String productName;

int quantity;

double price;

public Product(int id, String name, int qty, double price) {

this.productId = id;

this.productName = name;

this.quantity = qty;

this.price = price;

}

public String toString() {

return "[" + productId + "] " + productName + " | Qty: " + quantity + " | ₹" + price;

}

}

class InventoryManager {

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

public void addProduct(Product p) {

inventory.put(p.productId, p);

System.out.println("✅ Product added: " + p.productName);

}

public void updateProduct(int id, int qty, double price) {

if (inventory.containsKey(id)) {

Product p = inventory.get(id);

p.quantity = qty;

p.price = price;

System.out.println("🔄 Product updated: " + p.productName);

} else {

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

}

}

public void deleteProduct(int id) {

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

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

} else {

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

}

}

public void showInventory() {

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

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

System.out.println(p);

}

}

}

public class Main {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

manager.addProduct(new Product(1, "Mouse", 30, 250.50));

manager.addProduct(new Product(2, "Keyboard", 15, 550.00));

manager.showInventory();

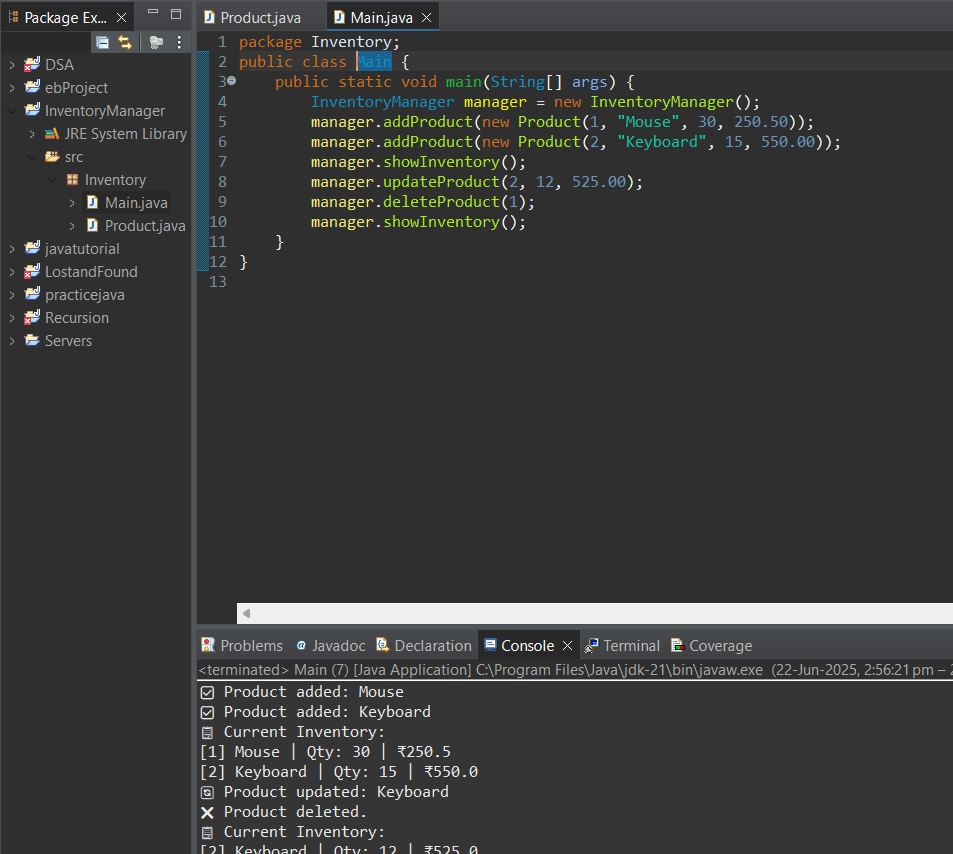
manager.updateProduct(2, 12, 525.00);

manager.deleteProduct(1);

manager.showInventory();

}}

**Output:**

****

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

**Code:**

package Ecomsearch;

import java.util.Arrays;

public class Prod {

int productId;

String productName;

String category;

public Prod(int id, String name, String cat) {

productId = id;

productName = name;

category = cat;

}

public String toString() {

return productName + " (" + category + ")";

}

}

class SearchEngine {

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

for (Product p : products) {

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

return p;

}

}

return null;

}

public static Prod binarySearch(Prod[] products, String key) {

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

while (left <= right) {

int mid = (left + right) / 2;

int compare = products[mid].productName.compareToIgnoreCase(key);

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

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

else right = mid - 1;

}

return null;

}

}

public class Main {

public static void main(String[] args) {

Prod[] products = {

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

new Prod(102, "Book", "Stationery"),

new Prod(103, "Camera", "Gadgets")

};

Arrays.sort(products, (a, b) -> a.productName.compareToIgnoreCase(b.productName));

System.out.println("🔎 Linear Search:");

Prod found1 = SearchEngine.linearSearch(products, "Camera");

System.out.println(found1 != null ? "✅ Found: " + found1 : "❌ Not Found");

System.out.println("🔎 Binary Search:");

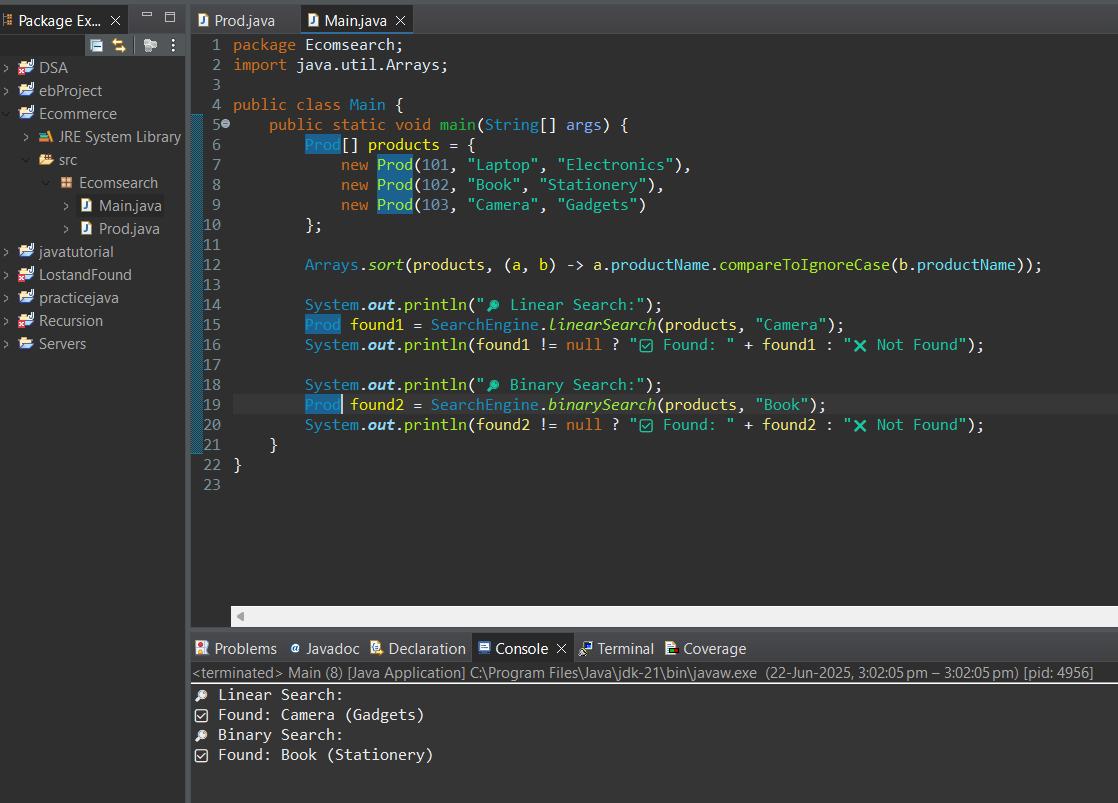
Prod found2 = SearchEngine.binarySearch(products, "Book");

System.out.println(found2 != null ? "✅ Found: " + found2 : "❌ Not Found");

}

}

**Output:**

****

**Exercise 3: Sorting Customer Orders**

**Code:**

package Sorting;

public class Order {

int orderId;

String customerName;

double totalPrice;

public Order(int id, String name, double price) {

orderId = id;

customerName = name;

totalPrice = price;

}

public String toString() {

return "#" + orderId + " - " + customerName + ": ₹" + totalPrice;

}

}

class Sorter {

public static void bubbleSort(Order[] orders) {

int n = orders.length;

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

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

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

Order temp = orders[j];

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

orders[j + 1] = temp;

}

}

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 class Main {

public static void main(String[] args) {

Order[] orders = {

new Order(1, "Amit", 450.0),

new Order(2, "Priya", 850.0),

new Order(3, "Rahul", 250.0)

};

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

for (Order o : orders) System.out.println(o);

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

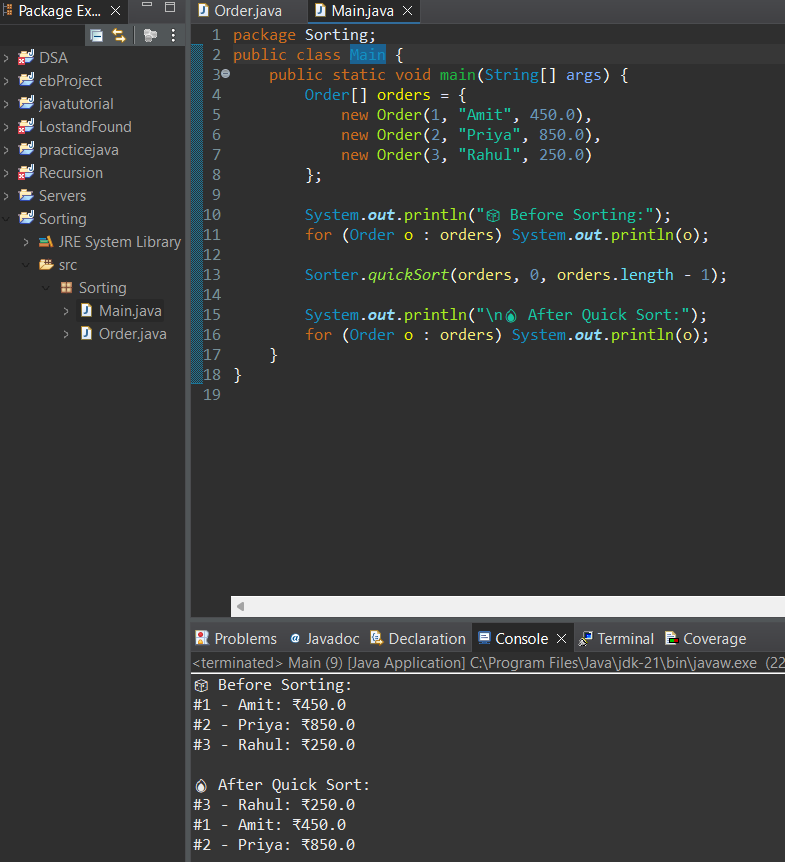
System.out.println("\n🔥 After Quick Sort:");

for (Order o : orders) System.out.println(o);

}

}

**Output:**

****

**Exercise 4: Employee Management(Arrays)**

**Code:**

Package Emp;

public class Employee {

int employeeId;

String name;

String position;

double salary;

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

this.employeeId = id;

this.name = name;

this.position = position;

this.salary = salary;

}

public String toString() {

return "#" + employeeId + " - " + name + " (" + position + ") ₹" + salary;

}

}

class EmployeeManager {

Employee[] employees = new Employee[10];

int count = 0;

public void addEmployee(Employee e) {

if (count < employees.length) {

employees[count++] = e;

System.out.println("✅ Employee added: " + e.name);

} else {

System.out.println("⚠️ Array full, cannot add more employees.");

}

}

public void showAll() {

System.out.println("📋 Employee List:");

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

System.out.println(employees[i]);

}

}

public void deleteEmployee(int id) {

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

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

System.out.println("❌ Deleting: " + employees[i].name);

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

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

}

employees[--count] = null;

return;

}

}

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

}

}

public class Main {

public static void main(String[] args) {

EmployeeManager manager = new EmployeeManager();

manager.addEmployee(new Employee(1, "Alice", "Developer", 55000));

manager.addEmployee(new Employee(2, "Bob", "Designer", 45000));

manager.showAll();

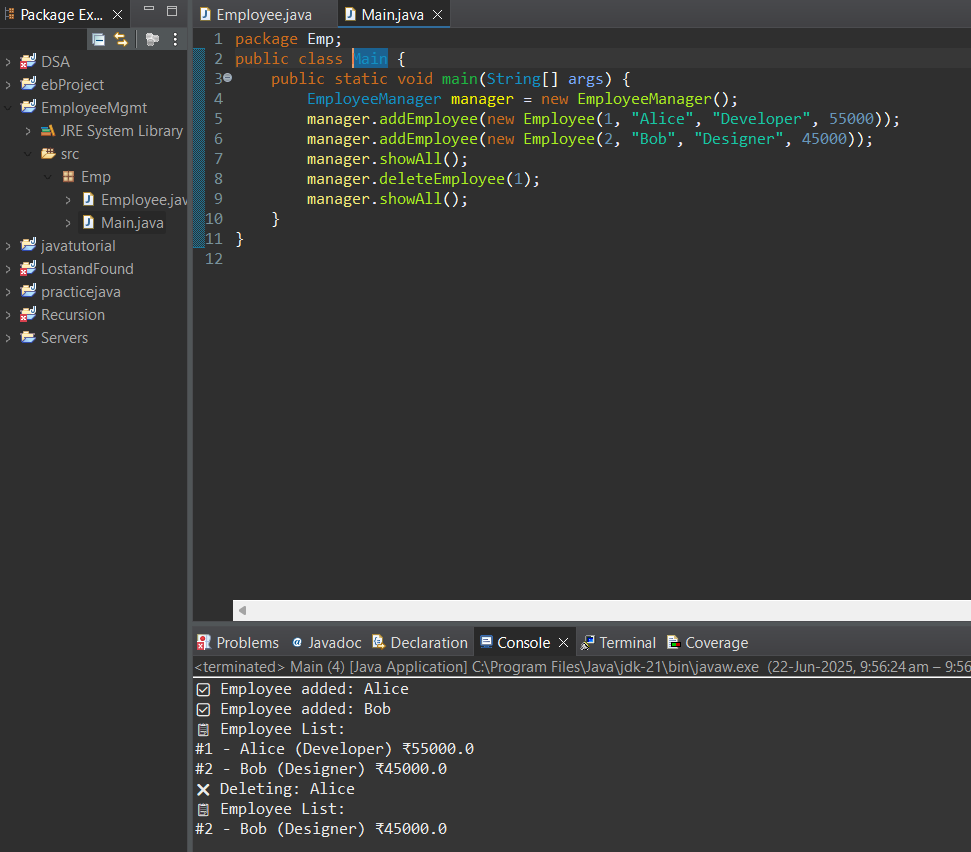
manager.deleteEmployee(1);

manager.showAll();

}

}

**Output:**

****

**Exercise 5: Task Management System(Singly Linked List)**

**Code:**

package Task;

public class Task {

int taskId;

String taskName;

String status;

Task next;

public Task(int id, String name, String status) {

this.taskId = id;

this.taskName = name;

this.status = status;

}

public String toString() {

return "[" + taskId + "] " + taskName + " - " + status;

}

}

class TaskList {

Task head;

public void addTask(Task newTask) {

if (head == null) {

head = newTask;

} else {

Task temp = head;

while (temp.next != null)

temp = temp.next;

temp.next = newTask;

}

System.***out***.println("📝 Task added: " + newTask.taskName);

}

public void showTasks() {

System.***out***.println("📋 Task List:");

Task temp = head;

while (temp != null) {

System.***out***.println(temp);

temp = temp.next;

}

}

public void deleteTask(int id) {

if (head == null) return;

if (head.taskId == id) {

System.***out***.println("❌ Deleting: " + head.taskName);

head = head.next;

return;

}

Task current = head;

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

current = current.next;

}

if (current.next != null) {

System.***out***.println("❌ Deleting: " + current.next.taskName);

current.next = current.next.next;

} else {

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

}

}

}

public class Main {

public static void main(String[] args) {

TaskList list = new TaskList();

list.addTask(new Task(101, "Create UI", "Pending"));

list.addTask(new Task(102, "Write API", "Ongoing"));

list.showTasks();

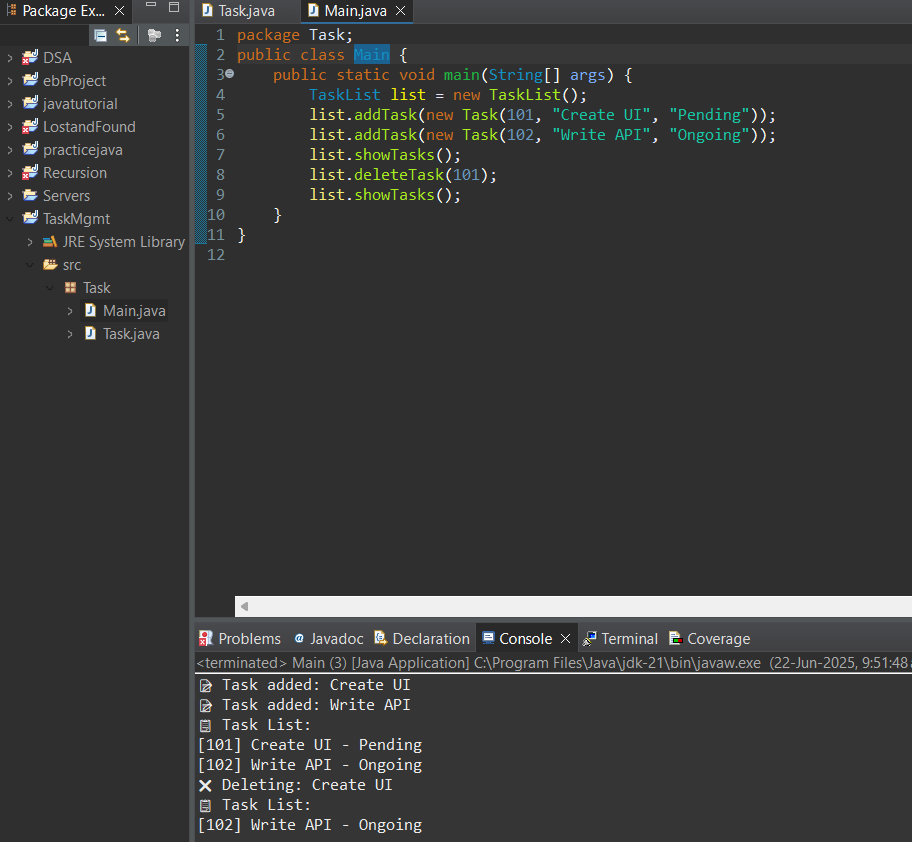
list.deleteTask(101);

list.showTasks();

}

}

**Output:**

****

**Exercise 6: Library Management System (Linear and Binary search)**

**Code:**

package Lib;

import java.util.Arrays;

public class Book {

int bookId;

String title;

String author;

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

this.bookId = id;

this.title = title;

this.author = author;

}

public String toString() {

return title + " by " + author;

}

}

class LibrarySearch {

public static Book linearSearch(Book[] books, String title) {

for (Book b : books) {

if (b.title.equalsIgnoreCase(title)) return b;

}

return null;

}

public static Book binarySearch(Book[] books, String title) {

int l = 0, r = books.length - 1;

while (l <= r) {

int mid = (l + r) / 2;

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

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

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

else r = mid - 1;

}

return null;

}

}

public class Main {

public static void main(String[] args) {

Book[] books = {

new Book(1, "Wings of Fire", "A.P.J. Abdul Kalam"),

new Book(2, "Rich Dad Poor Dad", "Robert Kiyosaki"),

new Book(3, "Clean Code", "Robert C. Martin")

};

Arrays.sort(books, (a, b) -> a.title.compareToIgnoreCase(b.title));

System.out.println("🔎 Linear Search for 'Clean Code':");

Book result1 = LibrarySearch.linearSearch(books, "Clean Code");

System.out.println(result1 != null ? "✅ Found: " + result1 : "❌ Not Found");

System.out.println("🔎 Binary Search for 'Wings of Fire':");

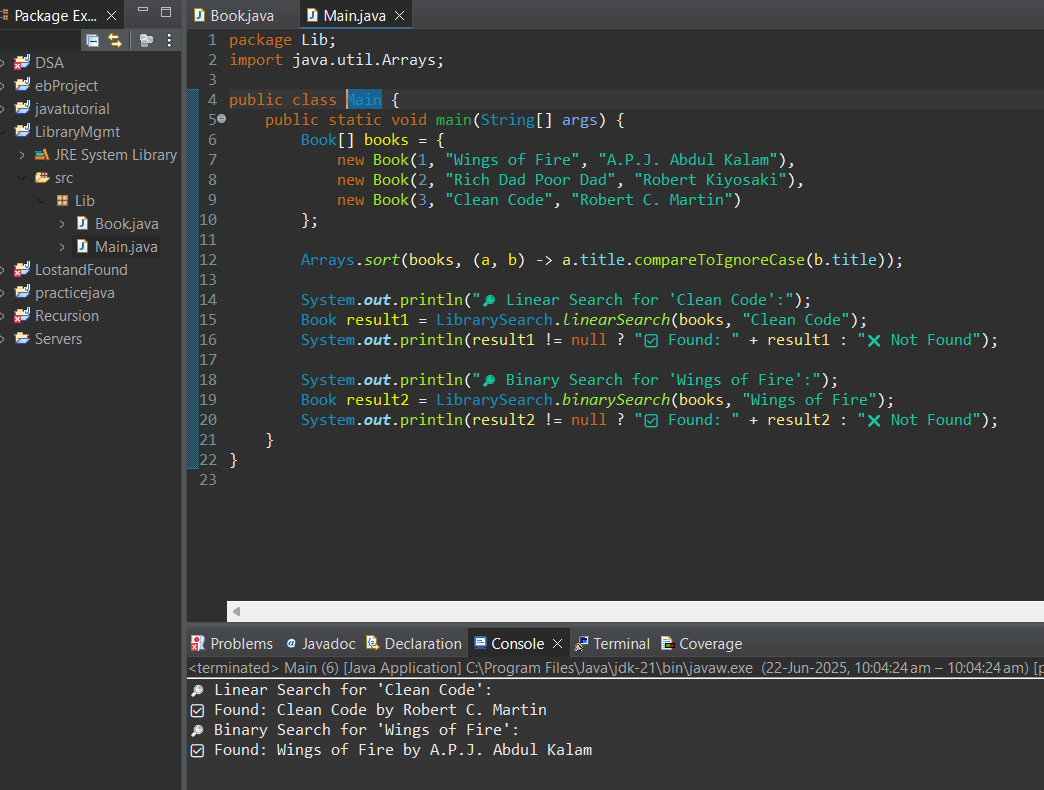
Book result2 = LibrarySearch.binarySearch(books, "Wings of Fire");

System.out.println(result2 != null ? "✅ Found: " + result2 : "❌ Not Found");

}

}

**Output:**

****

**Exercise 7: Financial Forecasting (Recursion)**

**Code:**

package Forecast;

public class Forecast {

public static double predict(double currentValue, double growthRate, int years) {

if (years == 0) return currentValue;

return predict(currentValue \* (1 + growthRate), growthRate, years - 1);

}

}

public class Main {

public static void main(String[] args) {

double initialValue = 10000;

double growthRate = 0.08;

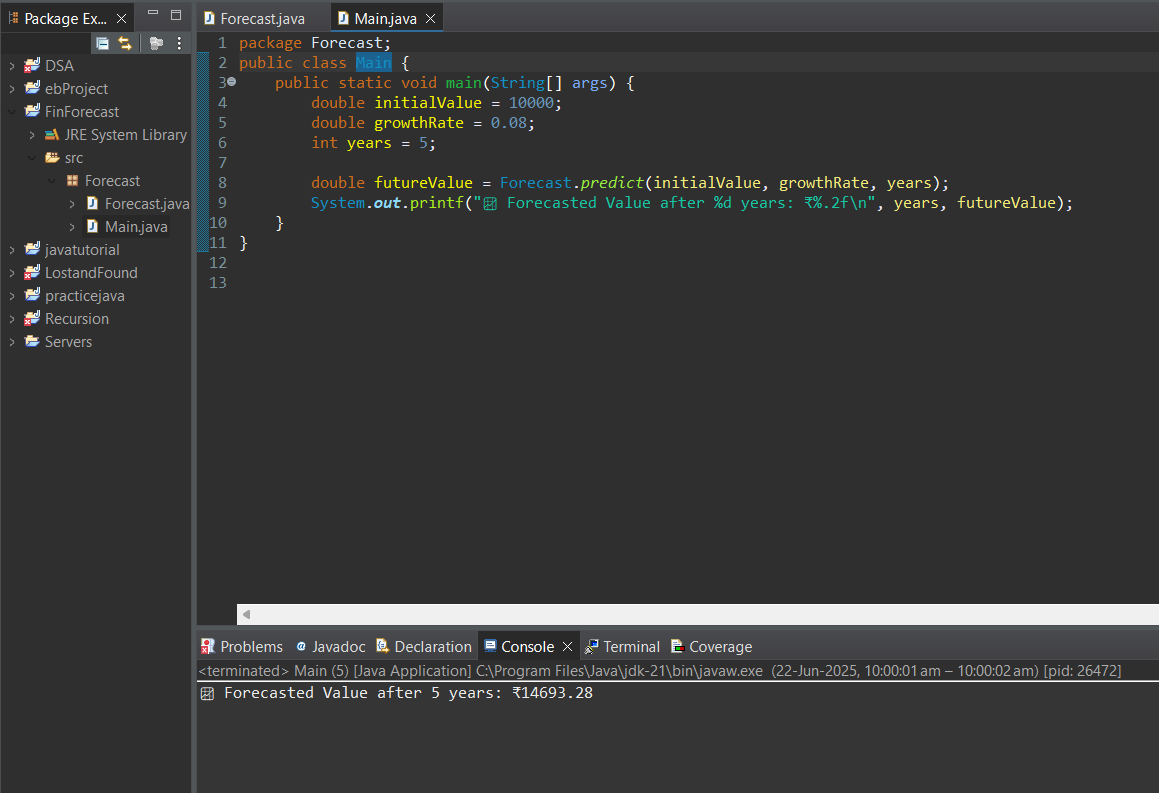
int years = 5;

double futureValue = Forecast.predict(initialValue, growthRate, years);

System.out.printf("📈 Forecasted Value after %d years: ₹%.2f\n", years, futureValue);

}}

**Output:**

****