**Algorithms and Data Structure Practice**

**(Yuvan Sakthi S)**

**1.Inventory Management System**

**Product.java**

**package** inventory;

**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 + ", Qty: " + quantity + ", Price: ₹" + price;

}

}

**Inventory.java**

**package** inventory;

**import** java.util.HashMap;

**public** **class** Inventory {

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

**public** **void** addProduct(Product product) {

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

System.***out***.println("Product added: " + product);

}

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

Product p = products.get(productId);

**if** (p != **null**) {

p.setQuantity(quantity);

p.setPrice(price);

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

} **else** {

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

}

}

**public** **void** deleteProduct(**int** productId) {

Product removed = products.remove(productId);

**if** (removed != **null**) {

System.***out***.println("Deleted: " + removed);

} **else** {

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

}

}

**public** **void** displayInventory() {

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

**for** (Product p : products.values()) {

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

}

}

}

TestInventory.java

**package** inventory;

**public** **class** TestInventory {

**public** **static** **void** main(String[] args) {

Inventory inventory = **new** Inventory();

Product p1 = **new** Product(101, "Laptop", 10, 85000);

Product p2 = **new** Product(102, "Mouse", 50, 400);

Product p3 = **new** Product(103, "Keyboard", 30, 1200);

inventory.addProduct(p1);

inventory.addProduct(p2);

inventory.addProduct(p3);

inventory.displayInventory();

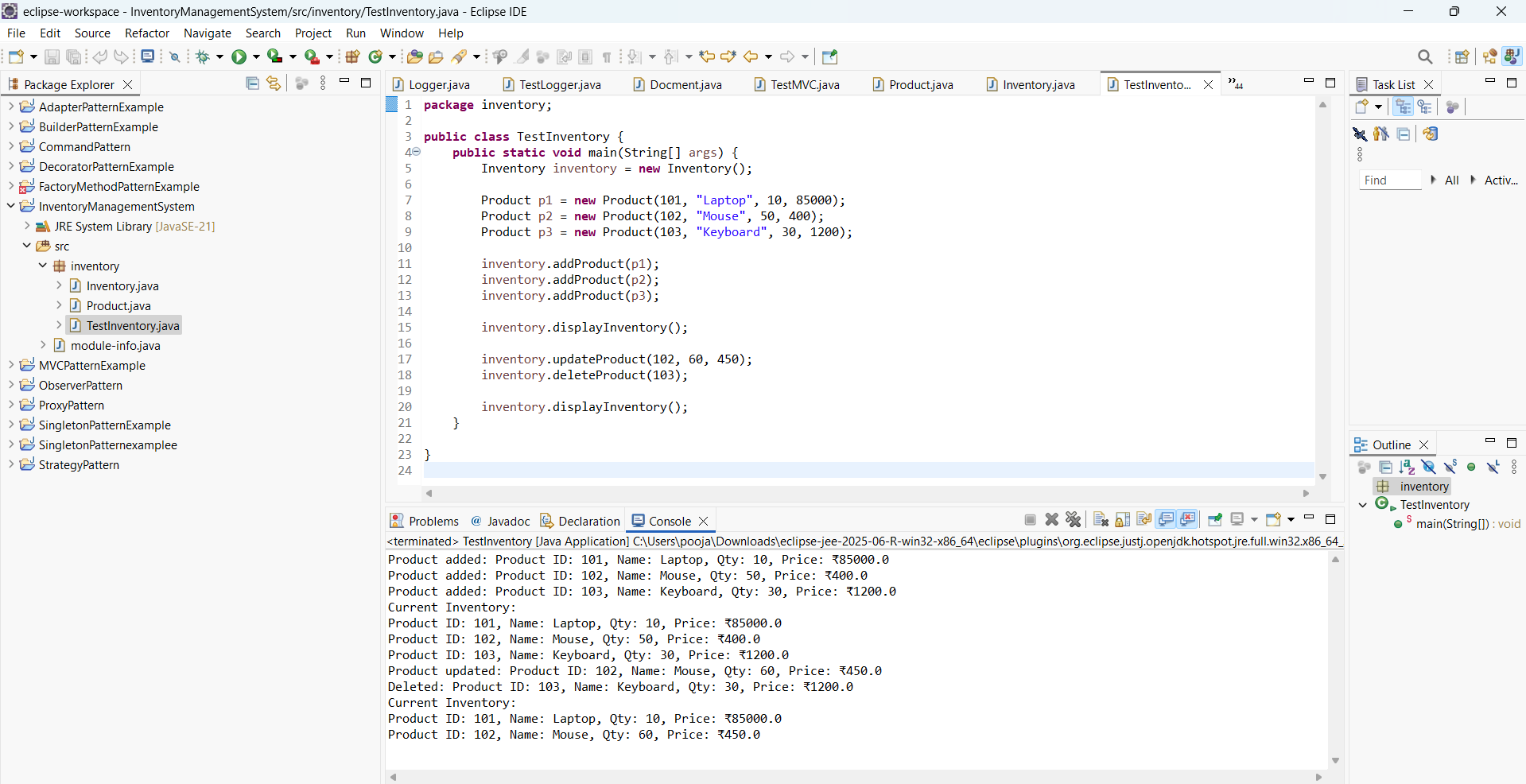
inventory.updateProduct(102, 60, 450);

inventory.deleteProduct(103);

inventory.displayInventory();

}

}



1.O/P

**2. E-commerce Platform Search Function**

**Product.java**

**package** ecommerce;

**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;

}

**public** String toString() {

**return** "[ + productId + "] "+productName + " - " + category;

}

}

**SearchEngine.java**

**package** ecommerce;

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** SearchEngine {

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

**for** (Product p : products) {

**if** (p.productName.equalsIgnoreCase(name)) {

**return** p;

}

}

**return** **null**;

}

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

**int** low = 0, high = products.length - 1;

**while** (low <= high) {

**int** mid = (low + high) / 2;

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

**if** (cmp == 0)

**return** products[mid];

**else** **if** (cmp < 0)

low = mid + 1;

**else**

high = mid - 1;

}

**return** **null**;

}

**public** **static** **void** sortByName(Product[] products) {

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

}

}

**TestSearch.java**

**package** ecommerce;

**public** **class** TestSearch {

**public** **static** **void** main(String[] args) {

Product[] products = {

**new** Product(101, "iPhone", "Electronics"),

**new** Product(102, "Samsung TV", "Electronics"),

**new** Product(103, "Nike Shoes", "Footwear"),

**new** Product(104, "Dell Laptop", "Computers"),

**new** Product(105, "Canon Camera", "Photography")

};

System.***out***.println("Linear Search for 'Nike Shoes':");

Product result1 = SearchEngine.*linearSearch*(products, "Nike Shoes");

System.***out***.println(result1 != **null** ? result1 : "Product not found.");

System.***out***.println("\n Binary Search for 'Dell Laptop':");

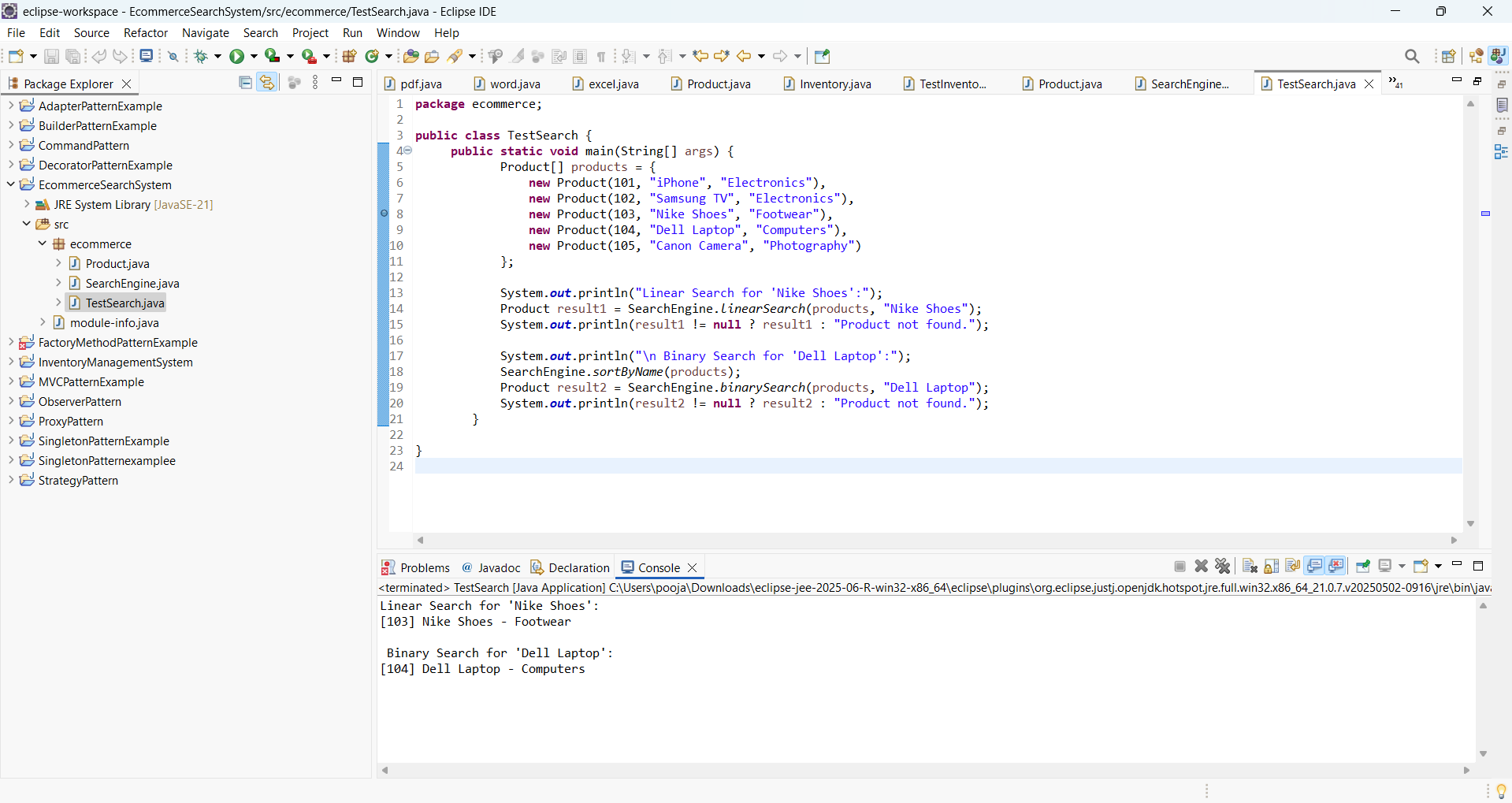
SearchEngine.*sortByName*(products);

Product result2 = SearchEngine.*binarySearch*(products, "Dell Laptop");

System.***out***.println(result2 != **null** ? result2 : "Product not found.");

}

}



2.O/P

**3.Sorting Customer Orders**

**Order.java:**

**package** sorting;

**public** **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;

}

@Override

**public** String toString() {

**return** "[" + orderId + "] " + customerName + " - ₹" + totalPrice;

}

}

**Sorter.java:**

**package** sorting;

**public** **class** Sorter {

**public** **static** **void** bubbleSort(Order[] orders) {

**int** n = orders.length;

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

**boolean** swapped = **false**;

**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;

swapped = **true**;

}

}

**if** (!swapped) **break**;

}

}

**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** printOrders(Order[] orders) {

**for** (Order o : orders) {

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

}

}

}

**TestSorting.java:**

**package** sorting;

**public** **class** TestSorting {

**public** **static** **void** main(String[] args) {

Order[] orders = {

**new** Order(301, "Akaash", 3500),

**new** Order(302, "Bhuvan", 1500),

**new** Order(303, "Chandru", 9000),

**new** Order(304, "Deepak", 4500),

**new** Order(305, "Evans Anto", 2500)

};

System.***out***.println("Original Orders:");

Sorter.*printOrders*(orders);

System.***out***.println("\n Sorted using Bubble Sort:");

Sorter.*bubbleSort*(orders);

Sorter.*printOrders*(orders);

orders = **new** Order[]{

**new** Order(301, "Akaash", 3500),

**new** Order(302, "Bhuvan", 1500),

**new** Order(303, "Chandru", 9000),

**new** Order(304, "Deepak", 4500),

**new** Order(305, "Evans Anto", 2500)

};

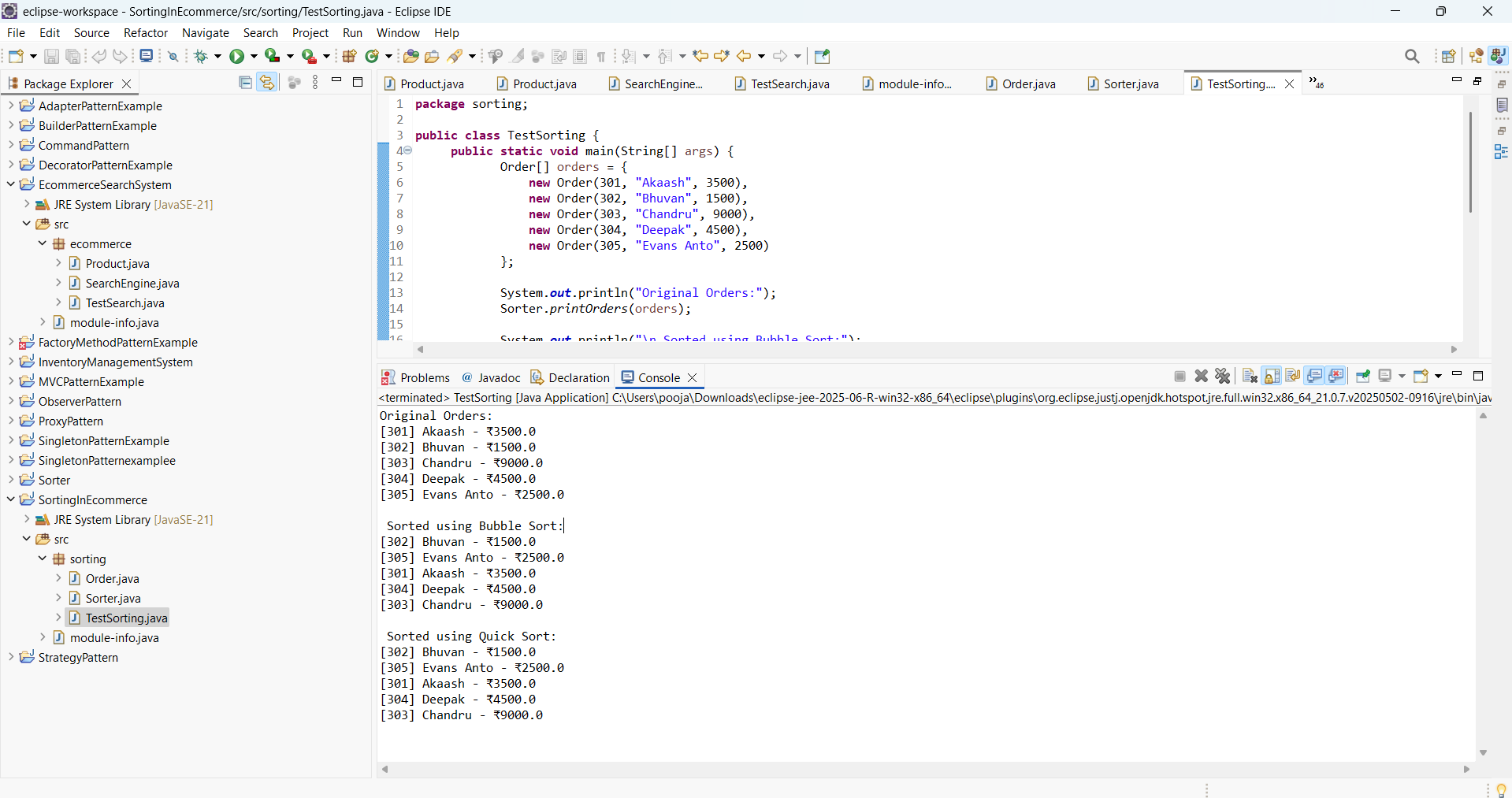
System.***out***.println("\n Sorted using Quick Sort:");

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

Sorter.*printOrders*(orders);

}

}



3.O/P

**4.Employee Management System**

**Employee.java:**

**package** empmgmtsystem;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

EmployeeManager manager = **new** EmployeeManager(5);

manager.addEmployee(**new** Employee(1, "Ajay", "Manager", 65000));

manager.addEmployee(**new** Employee(2, "Kaarthick", "Engineer", 55000));

manager.addEmployee(**new** Employee(3, "Rahul", "Analyst", 50000));

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

manager.displayAll();

System.***out***.println("\nSearching for ID 2:");

Employee found = manager.searchEmployee(2);

System.***out***.println(found != **null** ? found : "Not found");

System.***out***.println("\nDeleting ID 2:");

manager.deleteEmployee(2);

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

manager.displayAll();

}

}

**EmployeeManager.java:**

**package** empmgmtsystem;

**public** **class** EmployeeManager {

**private** Employee[] employees;

**private** **int** size;

**public** EmployeeManager(**int** capacity) {

employees = **new** Employee[capacity];

size = 0;

}

**public** **void** addEmployee(Employee e) {

**if** (size < employees.length) {

employees[size++] = e;

System.***out***.println("Added: " + e);

} **else** {

System.***out***.println("Array full! Cannot add more employees.");

}

}

**public** Employee searchEmployee(**int** empId) {

**for** (**int** i = 0; i < size; i++) {

**if** (employees[i].employeeId == empId) {

**return** employees[i];

}

}

**return** **null**;

}

**public** **void** displayAll() {

System.***out***.println("All Employees:");

**for** (**int** i = 0; i < size; i++) {

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

}

}

**public** **void** deleteEmployee(**int** empId) {

**for** (**int** i = 0; i < size; i++) {

**if** (employees[i].employeeId == empId) {

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

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

}

employees[size - 1] = **null**;

size--;

System.***out***.println("Deleted employee ID: " + empId);

**return**;

}

}

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

}

}

**Test.java:**

**package** empmgmtsystem;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

EmployeeManager manager = **new** EmployeeManager(5);

manager.addEmployee(**new** Employee(1, "Ajay", "Manager", 65000));

manager.addEmployee(**new** Employee(2, "Kaarthick", "Engineer", 55000));

manager.addEmployee(**new** Employee(3, "Rahul", "Analyst", 50000));

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

manager.displayAll();

System.***out***.println("\nSearching for ID 2:");

Employee found = manager.searchEmployee(2);

System.***out***.println(found != **null** ? found : "Not found");

System.***out***.println("\nDeleting ID 2:");

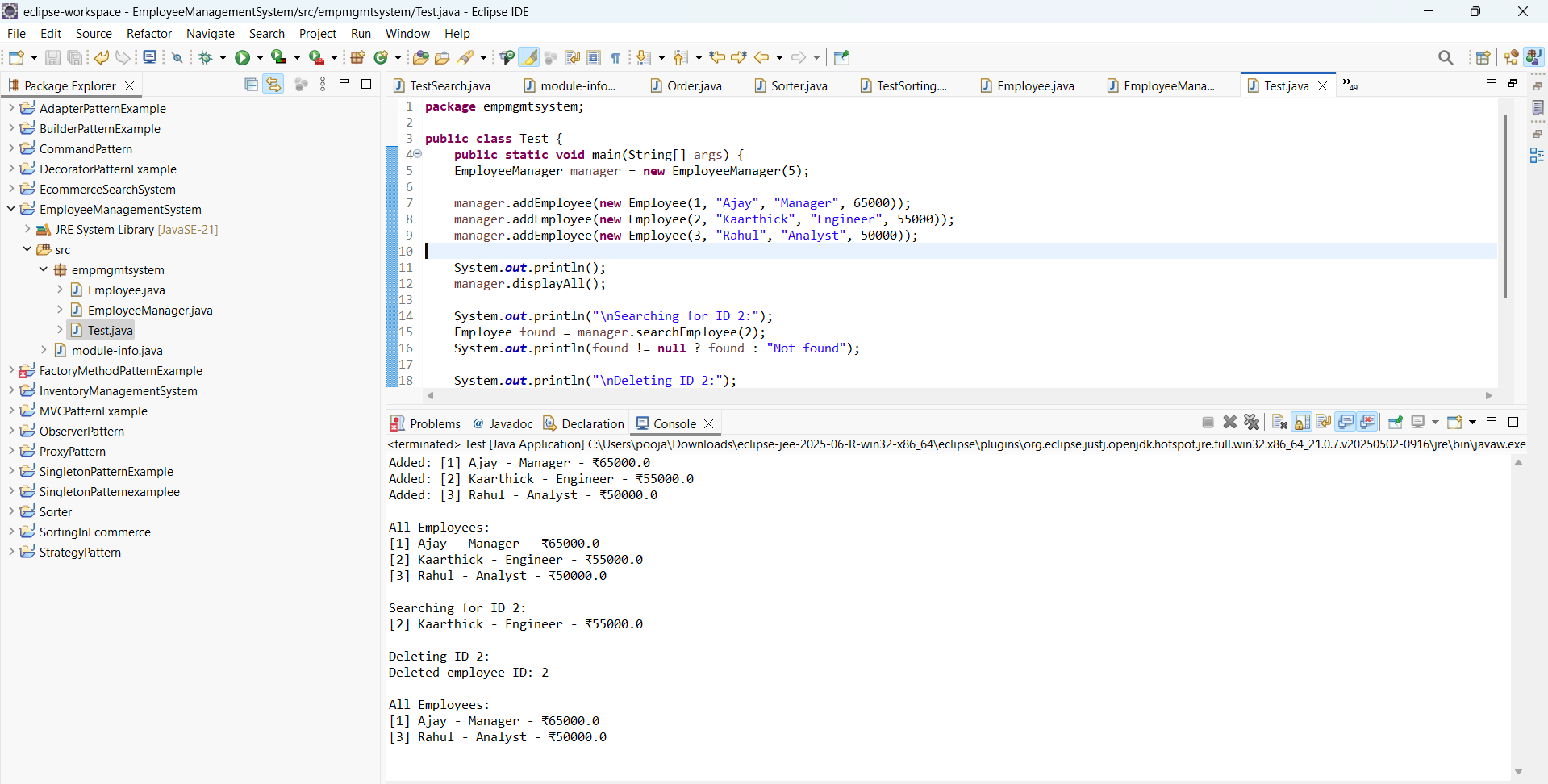
manager.deleteEmployee(2);

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

manager.displayAll();

}

}



4.O/P

**5. Task Management System**

**TaskNode.java:**

**package** taskManager;

**public** **class** TaskNode {

Task task;

TaskNode next;

**public** TaskNode(Task task) {

**this**.task=task;

**this**.next=**null**;

}

}

**Task.java:**

**package** taskManager;

**public** **class** Task {

**int** taskId;

String taskName;

String status;

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

**this**.taskId = taskId;

**this**.taskName = taskName;

**this**.status = status;

}

@Override

**public** String toString() {

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

}

}

**TaskManager.java:**

**package** taskManager;

**public** **class** TaskManager {

**private** TaskNode head;

**public** **void** addTask(Task task) {

TaskNode newNode=**new** TaskNode(task);

**if**(head==**null**) {

head=newNode;

}

**else** {

TaskNode temp=head;

**while**(temp.next!=**null**) {

temp=temp.next;

}

temp.next=newNode;

}

System.***out***.println("Task Added: " + task);

}

**public** **void** displayTasks() {

TaskNode temp = head;

**if** (temp == **null**) {

System.***out***.println("No tasks available.");

**return**;

}

System.***out***.println("All Tasks:");

**while** (temp != **null**) {

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

temp = temp.next;

}

}

**public** Task searchTask(**int** taskId) {

TaskNode temp = head;

**while** (temp != **null**) {

**if** (temp.task.taskId == taskId) {

**return** temp.task;

}

temp = temp.next;

}

**return** **null**;

}

**public** **void** deleteTask(**int** taskId) {

**if** (head == **null**) {

System.***out***.println("List is empty.");

**return**;

}

**if** (head.task.taskId == taskId) {

head = head.next;

System.***out***.println("Deleted task ID: " + taskId);

**return**;

}

TaskNode prev = head;

TaskNode curr = head.next;

**while** (curr != **null**) {

**if** (curr.task.taskId == taskId) {

prev.next = curr.next;

System.***out***.println("Deleted task ID: " + taskId);

**return**;

}

prev = curr;

curr = curr.next;

}

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

}

}

**TestTask.java:**

**package** taskManager;

**public** **class** TestTask {

**public** **static** **void** main(String[]args) {

TaskManager manager = **new** TaskManager();

manager.addTask(**new** Task(1, "Testing ", "Pending"));

manager.addTask(**new** Task(2, "Design ", "In Progress"));

manager.addTask(**new** Task(3, "Deploy", "Pending"));

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

manager.displayTasks();

System.***out***.println("\nSearching for task ID 2:");

Task t = manager.searchTask(2);

System.***out***.println(t != **null** ? t : "Not found");

System.***out***.println("\nDeleting task ID 1:");

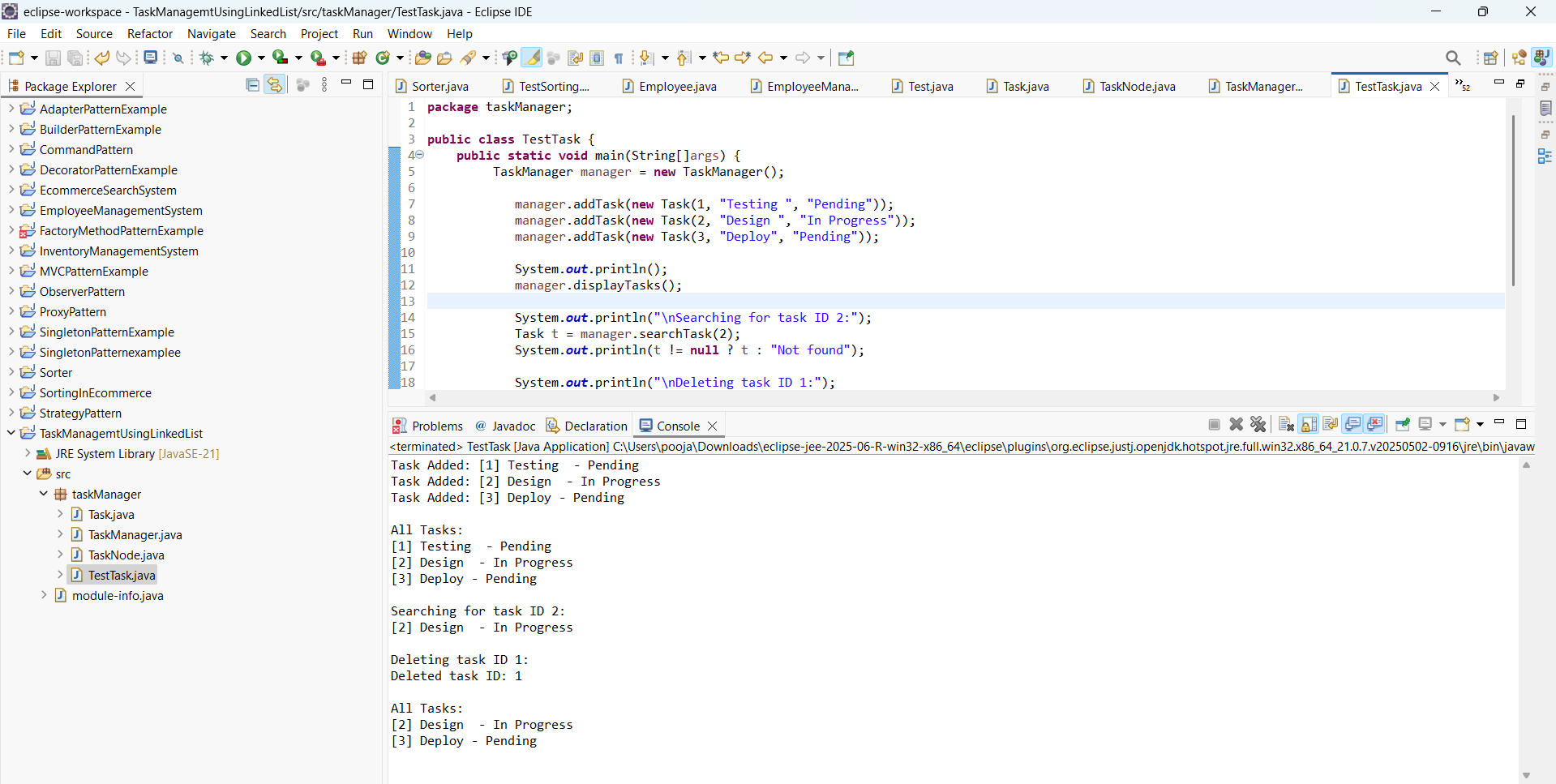
manager.deleteTask(1);

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

manager.displayTasks();

}

}



5.O/P

**6. Library Management System**

**Book.java**

**package** lib;

**public** **class** Book {

**int** bookId;

String title;

String author;

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

**this**.bookId = bookId;

**this**.title = title.toLowerCase();

**this**.author = author;

}

@Override

**public** String toString() {

**return** "[" + bookId + "] " + title + " by " + author;

}

}

**Librarymanager.java**

**package** lib;

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** LibraryManager {

Book[] books;

**int** size;

**public** LibraryManager(**int** capacity) {

books = **new** Book[capacity];

size = 0;

}

**public** **void** addBook(Book book) {

**if** (size < books.length) {

books[size++] = book;

} **else** {

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

}

}

**public** Book linearSearchByTitle(String title) {

**for** (**int** i = 0; i < size; i++) {

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

**return** books[i];

}

}

**return** **null**;

}

**public** Book binarySearchByTitle(String title) {

Arrays.*sort*(books, 0, size, Comparator.*comparing*(b -> b.title));

**int** low = 0, high = size - 1;

title = title.toLowerCase();

**while** (low <= high) {

**int** mid = (low + high) / 2;

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

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

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

**else** high = mid - 1;

}

**return** **null**;

}

**public** **void** displayAllBooks() {

System.***out***.println("Books in Library:");

**for** (**int** i = 0; i < size; i++) {

System.***out***.println(books[i]);

}

}

}

**TestLibrary.java:**

**package** lib;

**public** **class** TestLibrary {

**public** **static** **void** main(String[] args) {

LibraryManager manager = **new** LibraryManager(5);

manager.addBook(**new** Book(101, "Java Programming", "Jav"));

manager.addBook(**new** Book(102, "Data Structures", "DS"));

manager.addBook(**new** Book(103, "Python Basics", "Py"));

manager.addBook(**new** Book(104, "Operating Systems", "OSS"));

manager.addBook(**new** Book(105, "Algorithms", "Alg"));

manager.displayAllBooks();

System.***out***.println("\nLinear Search for 'Python Basics':");

Book b1 = manager.linearSearchByTitle("Python Basics");

System.***out***.println(b1 != **null** ? b1 : "Book not found.");

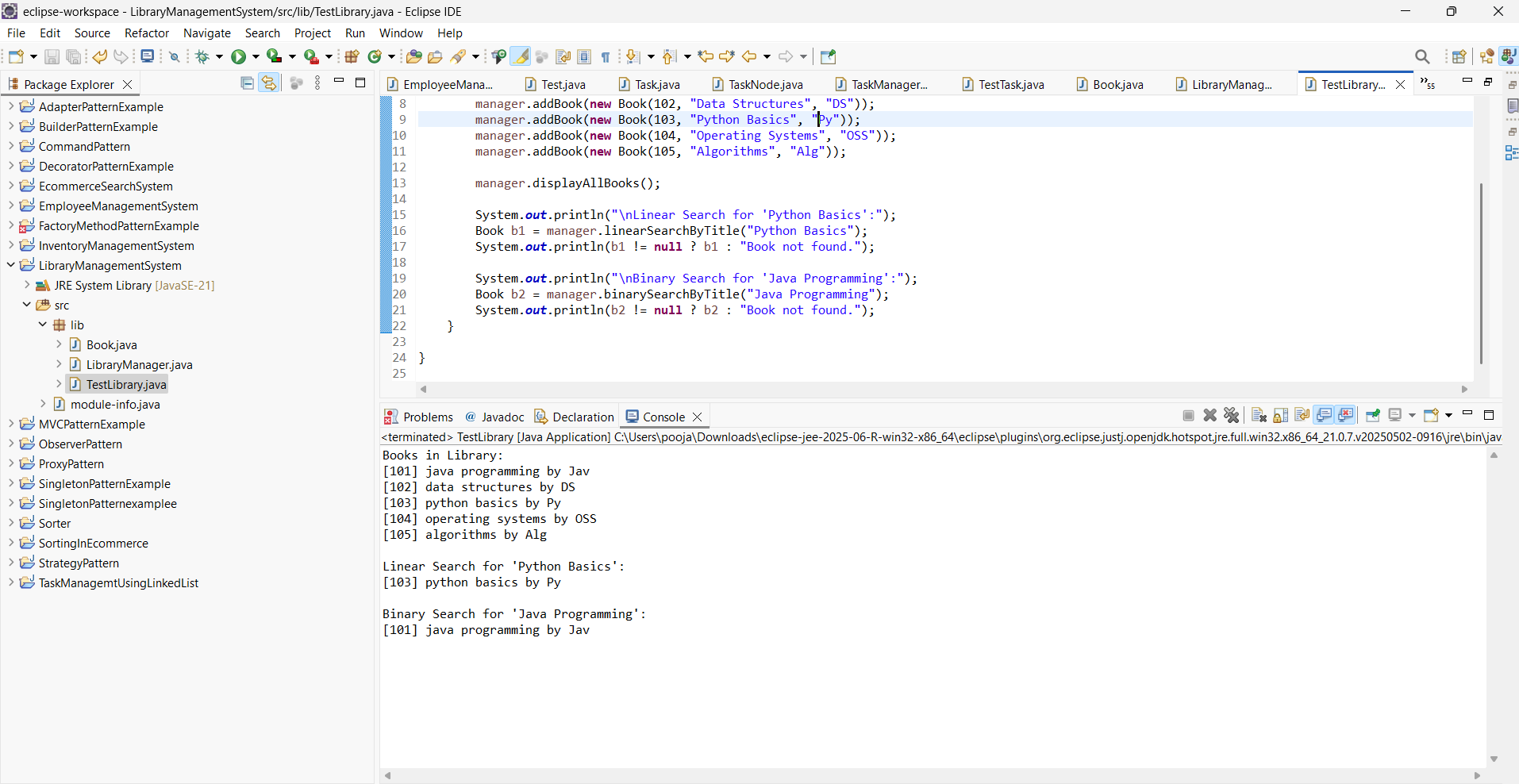
System.***out***.println("\nBinary Search for 'Java Programming':");

Book b2 = manager.binarySearchByTitle("Java Programming");

System.***out***.println(b2 != **null** ? b2 : "Book not found.");

}

}



6.O/P

**7: Financial Forecasting**

**FinancialForecaster.java:**

**package** forecast;

**import** java.util.HashMap;

**public** **class** FinancialForecaster {

**public** **double** forecastValueRecursive(**double** currentValue, **double** growthRate, **int** years) {

**if** (years == 0) {

**return** currentValue;

}

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

}

**private** HashMap<Integer, Double> m1 = **new** HashMap<>();

**public** **double** forecastValueMemo(**double** currentValue, **double** growthRate, **int** years) {

**if** (years == 0) **return** currentValue;

**if** (m1.containsKey(years)) **return** m1.get(years);

**double** value = forecastValueMemo(currentValue, growthRate, years - 1) \* (1 + growthRate);

m1.put(years, value);

**return** value;

}

**public** **double** forecastValueIterative(**double** currentValue, **double** growthRate, **int** years) {

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

currentValue \*= (1 + growthRate);

}

**return** currentValue;

}

}

**TestForecast.java:**

**package** forecast;

**public** **class** TestForecast {

**public** **static** **void** main(String[] args) {

FinancialForecaster forecaster = **new** FinancialForecaster();

**double** currentValue = 100000;

**double** growthRate = 0.08;

**int** years = 15;

System.***out***.println("Recursive Forecast: ₹" +

forecaster.forecastValueRecursive(currentValue, growthRate, years));

System.***out***.println("Memoized Forecast: ₹" +

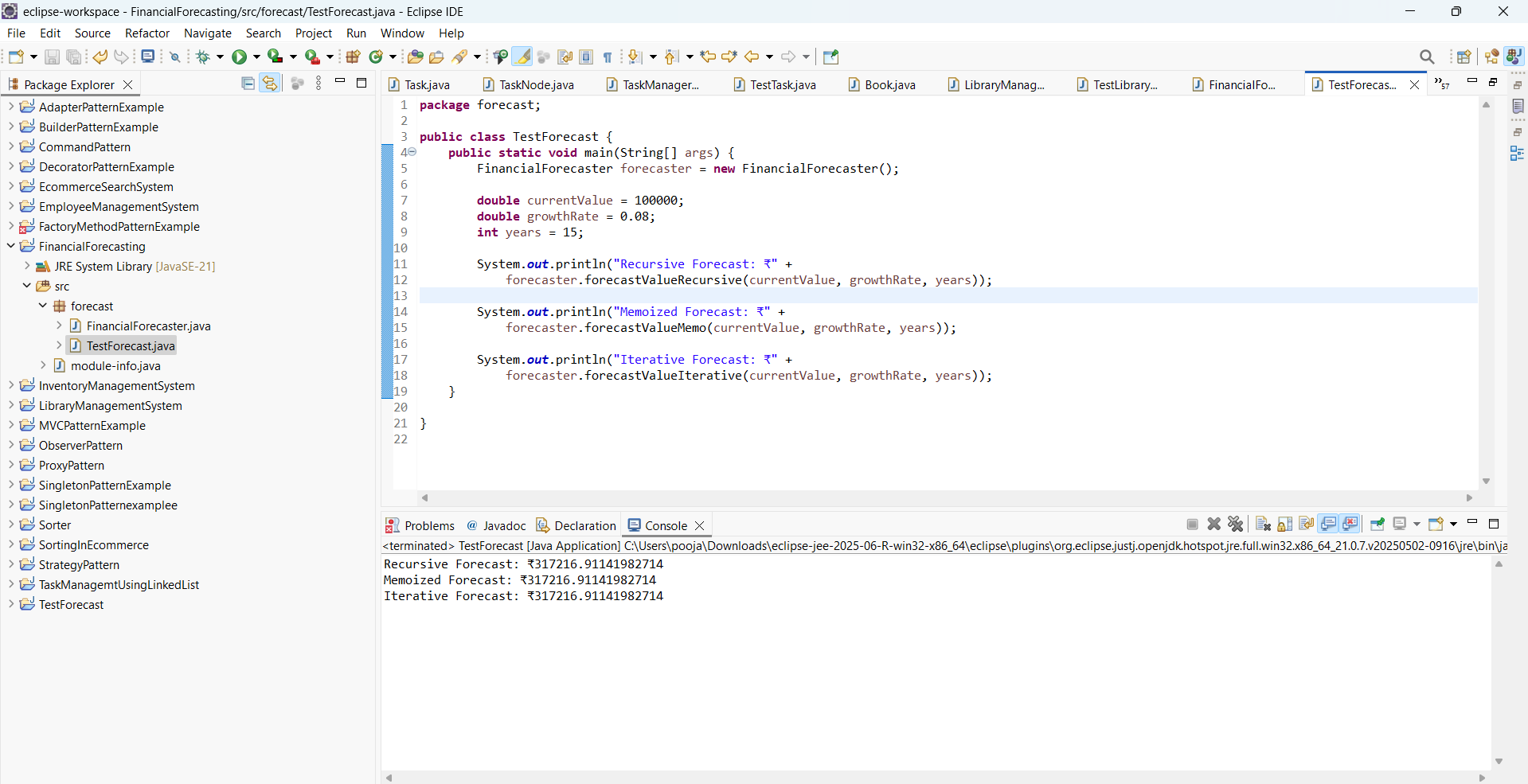
forecaster.forecastValueMemo(currentValue, growthRate, years));

System.***out***.println("Iterative Forecast: ₹" +

forecaster.forecastValueIterative(currentValue, growthRate, years));

}

}



7.O/P