**Exercise 1: Inventory Management System**

**CODE:**

**Product.java:**  
  
package com.inventory.product;

public 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;

}

@Override

public String toString() {

return "ID: " + productId + ", Name: " + productName + ", Quantity: " + quantity + ", Price: " + price;

}

}

**InventoryManager.java:**  
package com.inventory.product;

import java.util.HashMap;

public class InventoryManager {

private HashMap<Integer, Product> inventory;

public InventoryManager() {

inventory = new HashMap<>();

}

// Add a product

public void addProduct(Product product) {

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

System.out.println("Product already exists. Use update method.");

} else {

inventory.put(product.productId, product);

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

}

}

// Update a product

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

if (inventory.containsKey(productId)) {

Product p = inventory.get(productId);

p.productName = name;

p.quantity = quantity;

p.price = price;

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

} else {

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

}

}

// Delete a product

public void deleteProduct(int productId) {

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

System.out.println("Product removed.");

} else {

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

}

}

// View all products

public void viewInventory() {

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

System.out.println(p);

}

}

}

**Main.java:**

package com.inventory.product;

public class Main {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

Product p1 = new Product(101, "Laptop", 10, 999.99);

Product p2 = new Product(102, "Keyboard", 50, 19.99);

manager.addProduct(p1);

manager.addProduct(p2);

manager.viewInventory();

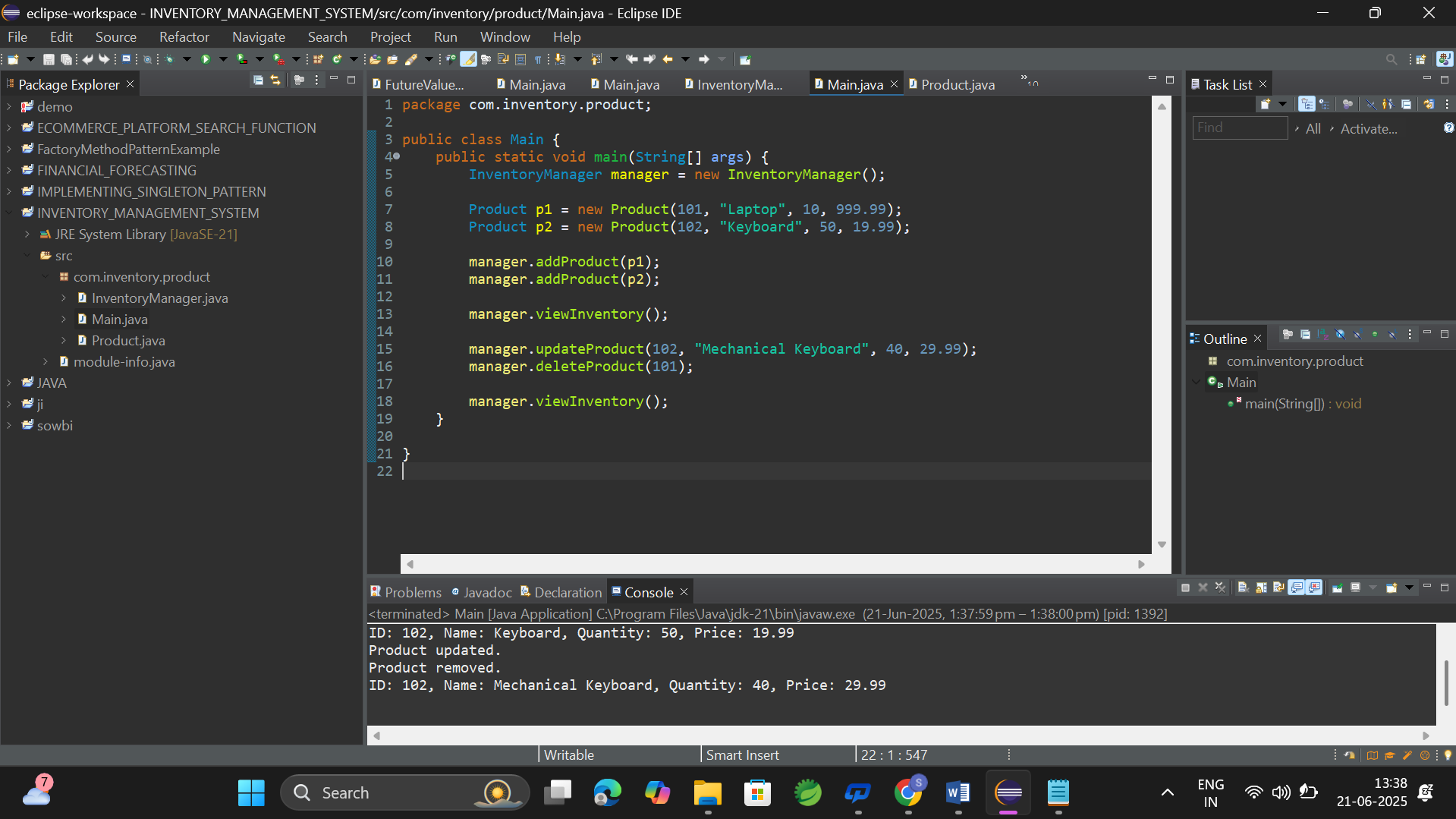
manager.updateProduct(102, "Mechanical Keyboard", 40, 29.99);

manager.deleteProduct(101);

manager.viewInventory();

}

}

**OUTPUT:**  


**Exercise 3: Sorting Customer Orders**

**CODE:**

**Order.java:**

public class Order {

private int orderId;

private String customerName;

private double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public double getTotalPrice() {

return totalPrice;

}

public String toString() {

return "OrderID: " + orderId + ", Name: " + customerName + ", Total: $" + totalPrice;

}

}

**BubbleSort.java:**

public class BubbleSort {

public static void sort(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].getTotalPrice() > orders[j + 1].getTotalPrice()) {

// swap

Order temp = orders[j];

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

orders[j + 1] = temp;

}

}

}

}

}

**QuickSort.java:**

public class QuickSort {

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

if (low < high) {

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

sort(orders, low, pi - 1);

sort(orders, pi + 1, high);

}

}

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

double pivot = orders[high].getTotalPrice();

int i = low - 1;

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

if (orders[j].getTotalPrice() < 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;

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

Order[] orders = {

new Order(101, "Alice", 250.0),

new Order(102, "Bob", 120.5),

new Order(103, "Charlie", 499.9),

new Order(104, "David", 75.0),

new Order(105, "Eva", 320.0)

};

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

printOrders(orders);

// Bubble Sort

Order[] bubbleSorted = orders.clone();

BubbleSort.sort(bubbleSorted);

System.out.println("\nBubble Sorted Orders:");

printOrders(bubbleSorted);

// Quick Sort

Order[] quickSorted = orders.clone();

QuickSort.sort(quickSorted, 0, quickSorted.length - 1);

System.out.println("\nQuick Sorted Orders:");

printOrders(quickSorted);

}

private static void printOrders(Order[] orders) {

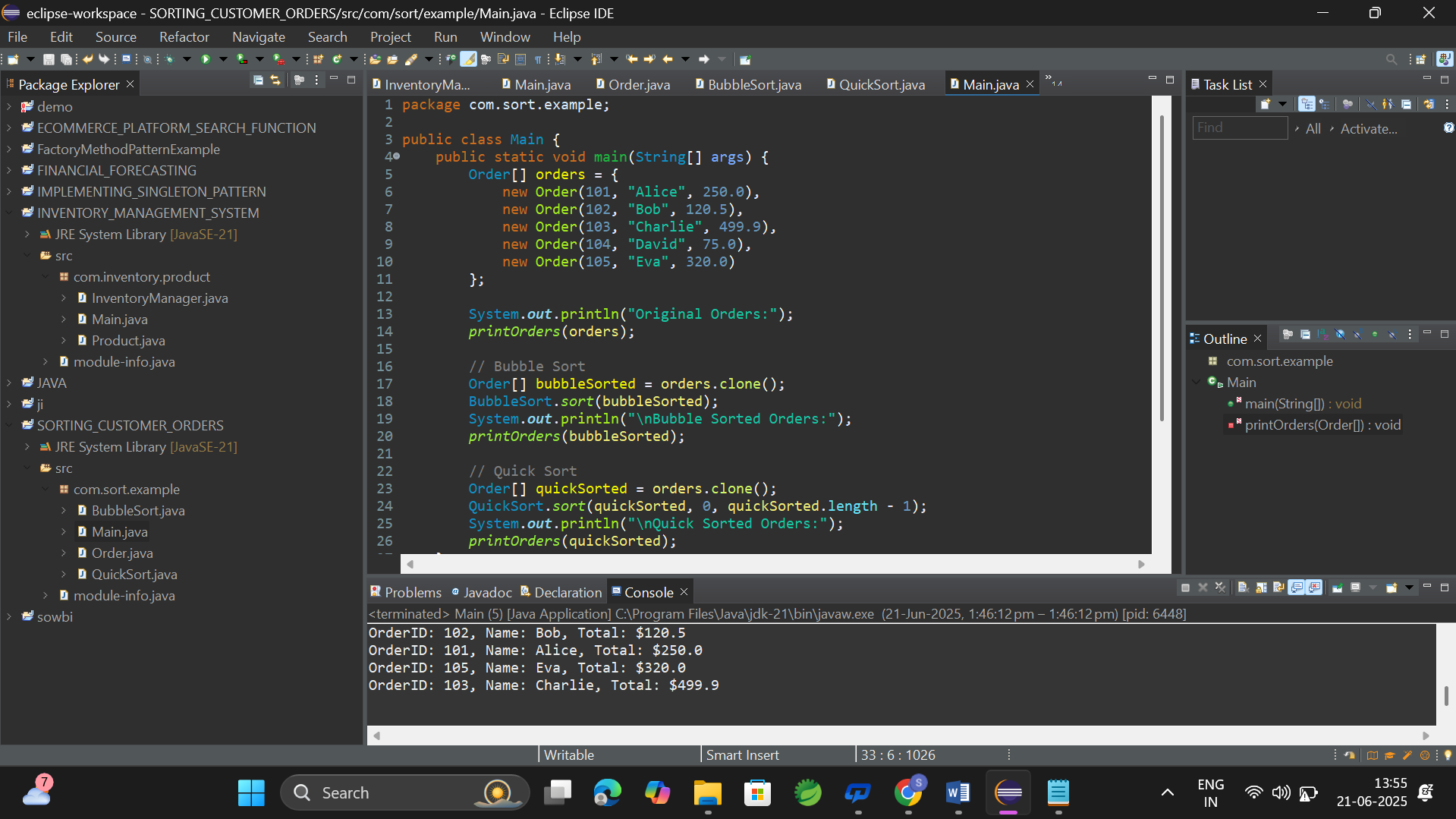
for (Order order : orders) {

System.out.println(order);

}

}

}

**Output:  
**

**Exercise 4: Employee Management System**

**Code:  
Employee.java:**

public 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);

}

}

**EmployeeManagementSystem.java:**

public class EmployeeManagementSystem {

private Employee[] employees;

private int count;

public EmployeeManagementSystem(int size) {

employees = new Employee[size];

count = 0;

}

// Add an employee

public void addEmployee(Employee emp) {

if (count < employees.length) {

employees[count++] = emp;

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

} else {

System.out.println("Cannot add employee: Array is full.");

}

}

// Search an employee by ID

public Employee searchEmployee(int empId) {

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

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

return employees[i];

}

}

return null;

}

// Traverse all employees

public void traverseEmployees() {

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

employees[i].display();

}

}

// Delete an employee by ID

public void deleteEmployee(int empId) {

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

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

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.");

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

EmployeeManagementSystem system = new EmployeeManagementSystem(5);

system.addEmployee(new Employee(101, "Alice", "Manager", 50000));

system.addEmployee(new Employee(102, "Bob", "Developer", 40000));

system.addEmployee(new Employee(103, "Charlie", "Designer", 42000));

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

system.traverseEmployees();

System.out.println("\nSearching for Employee ID 102:");

Employee emp = system.searchEmployee(102);

if (emp != null) emp.display();

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

System.out.println("\nDeleting Employee ID 102:");

system.deleteEmployee(102);

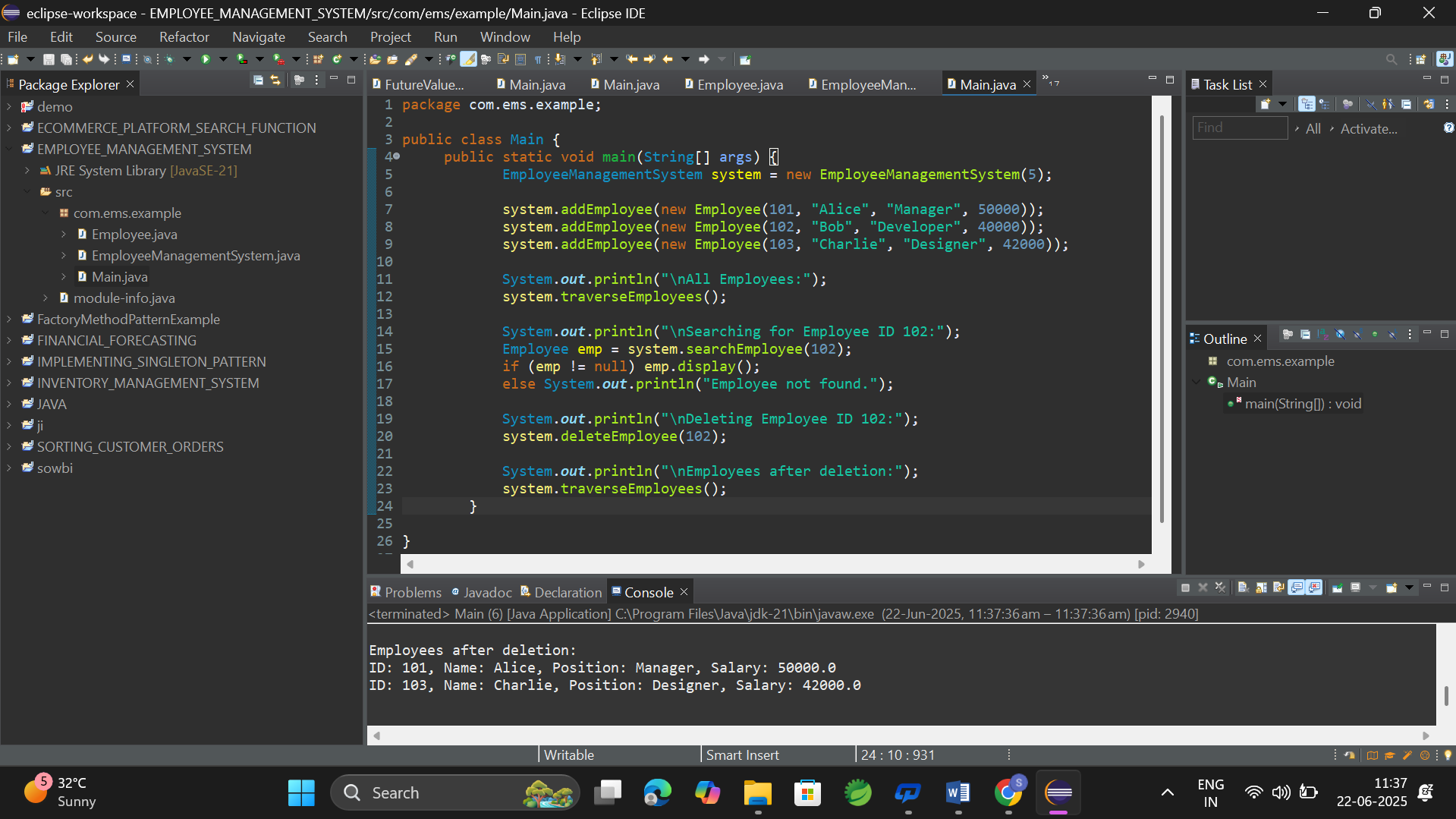
System.out.println("\nEmployees after deletion:");

system.traverseEmployees();

}

}

**OUTPUT:**



**Exercise 5: Task Management System**

**Code:**

**Task.java:**

public class Task {

int taskId;

String taskName;

String status;

Task next; // pointer to next Task node

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("Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status);

}

}

**TaskManager.java:**

public class TaskManager {

private Task head;

public TaskManager() {

head = null;

}

// Add task at end

public void addTask(int taskId, String taskName, String status) {

Task newTask = new Task(taskId, taskName, status);

if (head == null) {

head = newTask;

} else {

Task current = head;

while (current.next != null) {

current = current.next;

}

current.next = newTask;

}

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

}

// Traverse all tasks

public void traverseTasks() {

if (head == null) {

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

return;

}

Task current = head;

while (current != null) {

current.display();

current = current.next;

}

}

// Search task by ID

public Task searchTask(int taskId) {

Task current = head;

while (current != null) {

if (current.taskId == taskId) {

return current;

}

current = current.next;

}

return null;

}

// Delete task by ID

public void deleteTask(int taskId) {

if (head == null) {

System.out.println("Task list is empty.");

return;

}

// if head is the one to delete

if (head.taskId == taskId) {

head = head.next;

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

return;

}

Task current = head;

Task prev = null;

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

prev = current;

current = current.next;

}

if (current == null) {

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

return;

}

prev.next = current.next;

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

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

TaskManager manager = new TaskManager();

manager.addTask(1, "Design UI", "Pending");

manager.addTask(2, "Build Backend", "In Progress");

manager.addTask(3, "Write Tests", "Pending");

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

manager.traverseTasks();

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

Task task = manager.searchTask(2);

if (task != null) task.display();

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

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

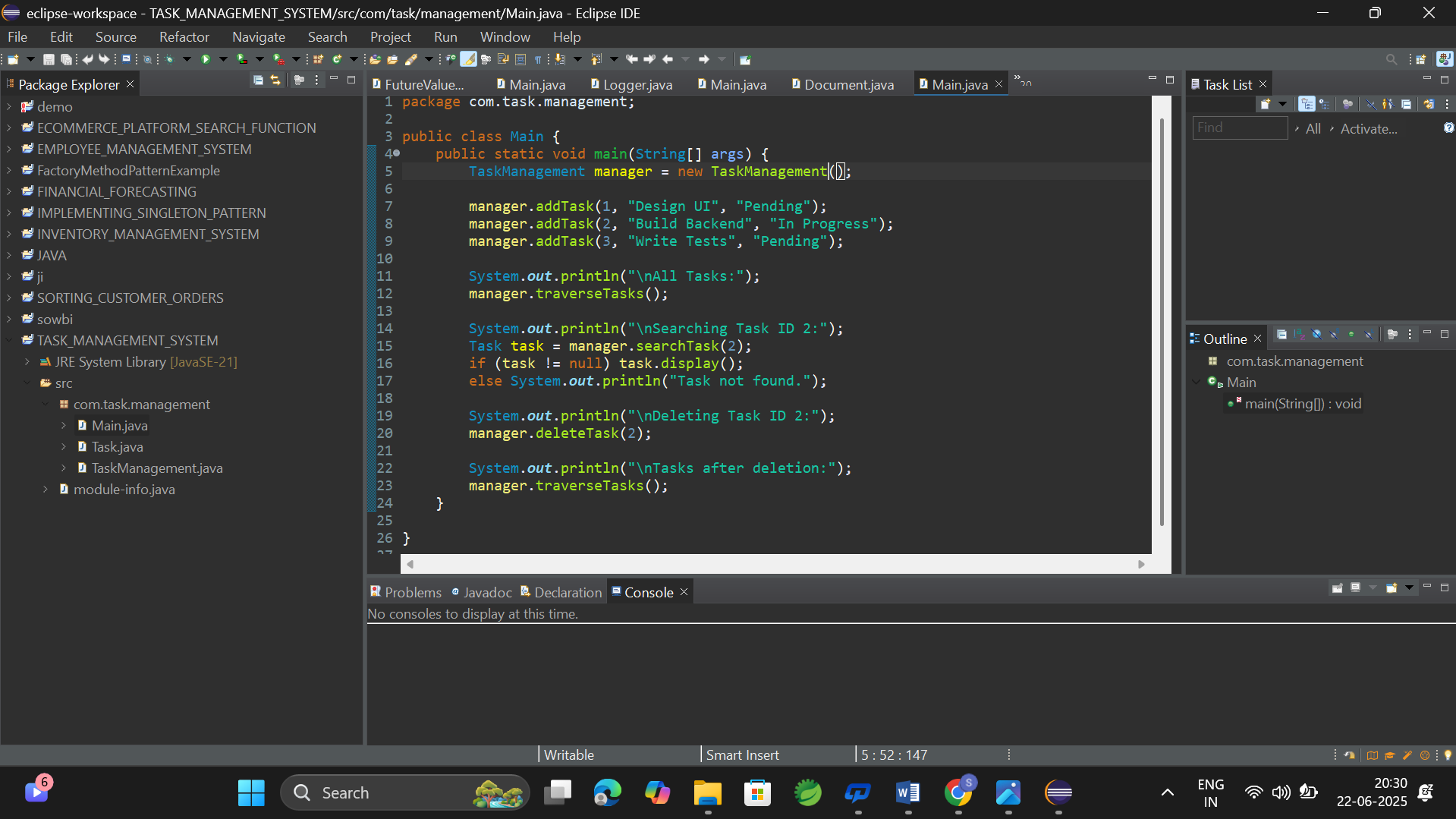
manager.deleteTask(2);

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

manager.traverseTasks();

}

}  
  
**Output:**



**Exercise 6: Library Management System**

**Code:**

**Book.java:**

public 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("Book ID: " + bookId + ", Title: " + title + ", Author: " + author);

}

}

**LibraryLinearSearch.java:**

public class LibraryLinearSearch {

private Book[] books;

private int count;

public LibraryLinearSearch(int size) {

books = new Book[size];

count = 0;

}

public void addBook(Book book) {

if (count < books.length) {

books[count++] = book;

} else {

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

}

}

public Book linearSearchByTitle(String title) {

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

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

return books[i];

}

}

return null;

}

public void displayAllBooks() {

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

books[i].display();

}

}

}

**LibraryBinarySearch.java:**

import java.util.Arrays;

import java.util.Comparator;

public class LibraryBinarySearch {

private Book[] books;

private int count;

public LibraryBinarySearch(int size) {

books = new Book[size];

count = 0;

}

public void addBook(Book book) {

if (count < books.length) {

books[count++] = book;

} else {

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

}

}

public Book binarySearchByTitle(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 = books[mid].title.compareToIgnoreCase(title);

if (cmp == 0) {

return books[mid];

} else if (cmp < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public void displayAllBooks() {

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

books[i].display();

}

}

}

**Main.java:**

public class Main {

public static void main(String[] args) {

// Linear Search Test

System.out.println("=== LINEAR SEARCH ===");

LibraryLinearSearch linearLibrary = new LibraryLinearSearch(5);

linearLibrary.addBook(new Book(1, "Java Basics", "James"));

linearLibrary.addBook(new Book(2, "Python 101", "Guido"));

linearLibrary.addBook(new Book(3, "C++ Primer", "Bjarne"));

linearLibrary.addBook(new Book(4, "HTML Guide", "Tim"));

linearLibrary.addBook(new Book(5, "Data Structures", "Seymour"));

linearLibrary.displayAllBooks();

System.out.println("\nSearching 'Python 101':");

Book result1 = linearLibrary.linearSearchByTitle("Python 101");

if (result1 != null) result1.display();

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

// Binary Search Test

System.out.println("\n=== BINARY SEARCH ===");

LibraryBinarySearch binaryLibrary = new LibraryBinarySearch(5);

binaryLibrary.addBook(new Book(1, "Java Basics", "James"));

binaryLibrary.addBook(new Book(2, "Python 101", "Guido"));

binaryLibrary.addBook(new Book(3, "C++ Primer", "Bjarne"));

binaryLibrary.addBook(new Book(4, "HTML Guide", "Tim"));

binaryLibrary.addBook(new Book(5, "Data Structures", "Seymour"));

binaryLibrary.displayAllBooks();

System.out.println("\nSearching 'C++ Primer':");

Book result2 = binaryLibrary.binarySearchByTitle("C++ Primer");

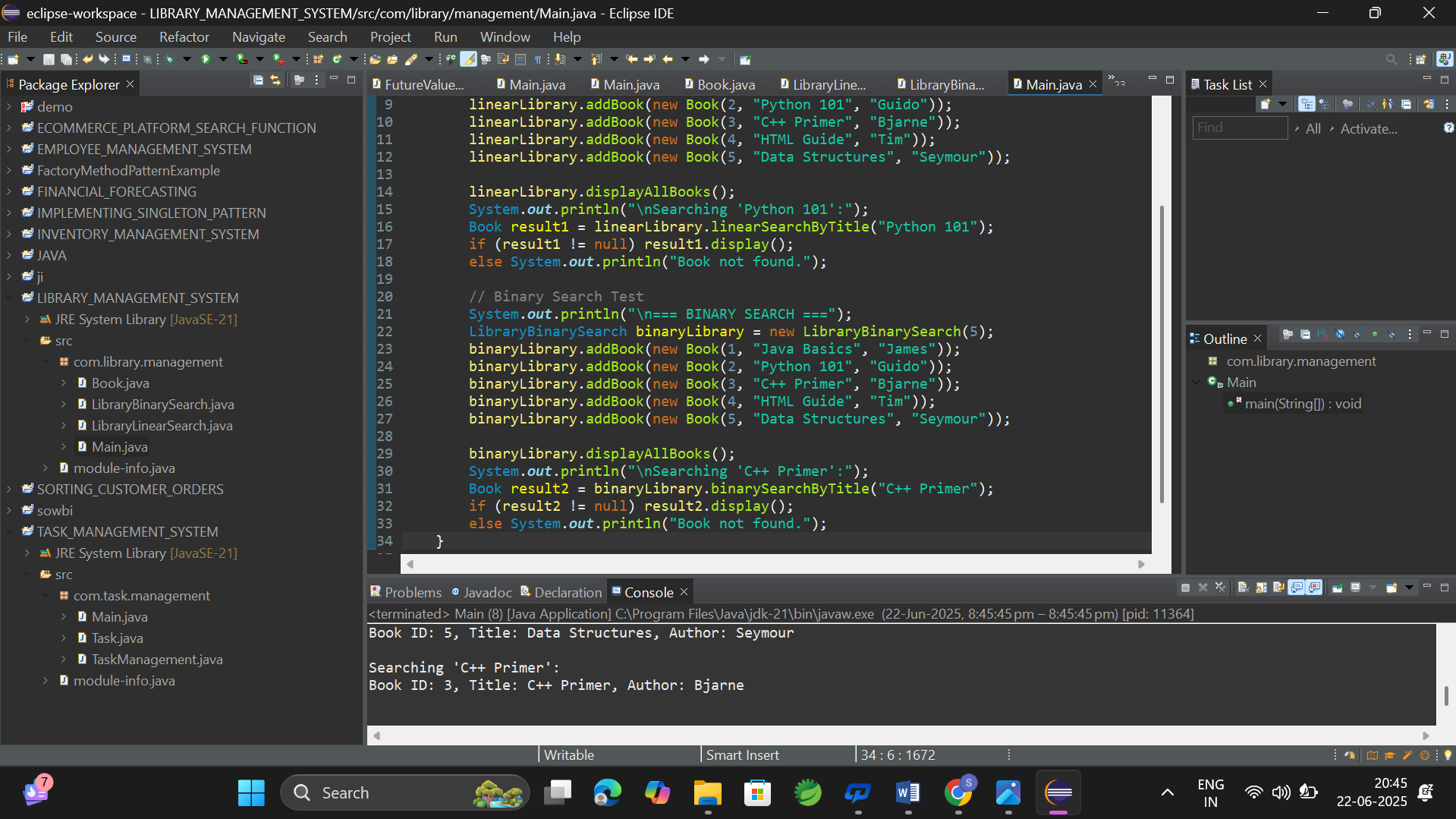
if (result2 != null) result2.display();

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

}

}

**Output:**

****