**DESIGN PATTERNS AND PRINCIPLES**

**Exercise 1: Implementing the Singleton Pattern**

// File: Logger.java

public class Logger {

// Private static instance of the same class

private static Logger instance;

// Private constructor to prevent instantiation

private Logger() {

System.out.println("Logger initialized.");

}

// Public static method to get the singleton instance

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

// Logging method

public void log(String message) {

System.out.println("[LOG] " + message);

}

}

// File: LoggerTest.java

public class LoggerTest {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("Starting application...");

Logger logger2 = Logger.getInstance();

logger2.log("Continuing application...");

// Validate both instances are the same

if (logger1 == logger2) {

System.out.println("Both logger instances are the same (singleton confirmed).");

} else {

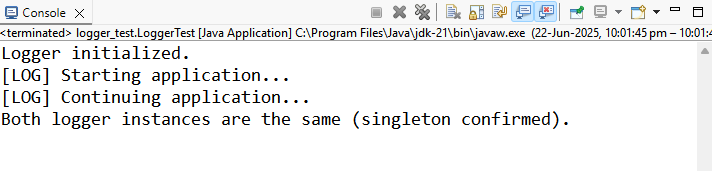
System.out.println("Different logger instances (singleton failed).");

}

}

}

OUTPUT:



**Exercise 2: Implementing the Factory Method Pattern**

// File: Document.java

public interface Document {

void open();

}

// File: WordDocument.java

public class WordDocument implements Document {

@Override

public void open() {

System.out.println("Opening Word document...");

}

}

// File: PdfDocument.java

public class PdfDocument implements Document {

@Override

public void open() {

System.out.println("Opening PDF document...");

}

}

// File: ExcelDocument.java

public class ExcelDocument implements Document {

@Override

public void open() {

System.out.println("Opening Excel document...");

}

}

// File: DocumentFactory.java

public abstract class DocumentFactory {

public abstract Document createDocument();

}

// File: WordDocumentFactory.java

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

// File: PdfDocumentFactory.java

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

// File: ExcelDocumentFactory.java

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

// File: DocumentFactoryTest.java

public class DocumentFactoryTest {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

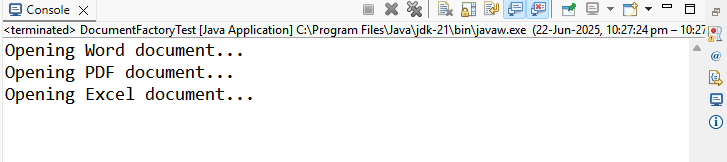
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

OUTPUT:



**Exercise 3: Implementing the Builder Pattern**

// File: Computer.java

public class Computer {

private final String cpu;

private final String ram;

private final String storage;

private final String gpu;

private final boolean isWifiEnabled;

private final boolean isBluetoothEnabled;

// Private constructor receives Builder

private Computer(Builder builder) {

this.cpu = builder.cpu;

this.ram = builder.ram;

this.storage = builder.storage;

this.gpu = builder.gpu;

this.isWifiEnabled = builder.isWifiEnabled;

this.isBluetoothEnabled = builder.isBluetoothEnabled;

}

// Static nested Builder class

public static class Builder {

private String cpu;

private String ram;

private String storage;

private String gpu;

private boolean isWifiEnabled;

private boolean isBluetoothEnabled;

public Builder setCpu(String cpu) {

this.cpu = cpu;

return this;

}

public Builder setRam(String ram) {

this.ram = ram;

return this;

}

public Builder setStorage(String storage) {

this.storage = storage;

return this;

}

public Builder setGpu(String gpu) {

this.gpu = gpu;

return this;

}

public Builder setWifiEnabled(boolean isWifiEnabled) {

this.isWifiEnabled = isWifiEnabled;

return this;

}

public Builder setBluetoothEnabled(boolean isBluetoothEnabled) {

this.isBluetoothEnabled = isBluetoothEnabled;

return this;

}

public Computer build() {

return new Computer(this);

}

}

public void displaySpecs() {

System.out.println("Computer Specs:");

System.out.println("CPU: " + cpu);

System.out.println("RAM: " + ram);

System.out.println("Storage: " + storage);

System.out.println("GPU: " + gpu);

System.out.println("WiFi: " + isWifiEnabled);

System.out.println("Bluetooth: " + isBluetoothEnabled);

}

}

// File: ComputerBuilderTest.java

public class ComputerBuilderTest {

public static void main(String[] args) {

Computer gamingPC = new Computer.Builder()

.setCpu("Intel i9")

.setRam("32GB")

.setStorage("1TB SSD")

.setGpu("NVIDIA RTX 4080")

.setWifiEnabled(true)

.setBluetoothEnabled(true)

.build();

Computer officePC = new Computer.Builder()

.setCpu("Intel i5")

.setRam("16GB")

.setStorage("512GB SSD")

.setWifiEnabled(true)

.build();

System.out.println("Gaming PC:");

gamingPC.displaySpecs();

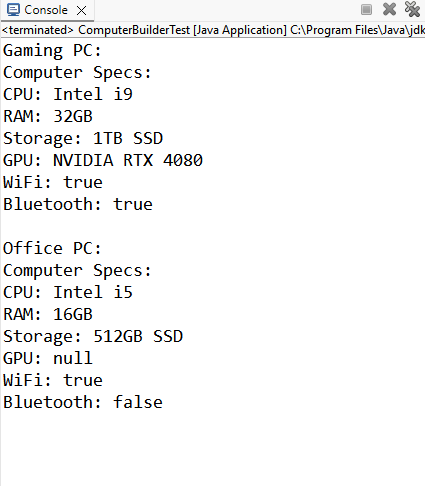
System.out.println("\nOffice PC:");

officePC.displaySpecs();

}

}

OUTPUT:



**Exercise 4: Implementing the Adapter Pattern**

// File: PaymentProcessor.java

public interface PaymentProcessor {

void processPayment(double amount);

}

// File: PayPalGateway.java

public class PayPalGateway {

public void makePayment(double amountInDollars) {

System.out.println("Processed payment via PayPal: $" + amountInDollars);

}

}

// File: StripeGateway.java

public class StripeGateway {

public void pay(double amount) {

System.out.println("Processed payment via Stripe: $" + amount);

}

}

// File: SquareGateway.java

public class SquareGateway {

public void doTransaction(double amt) {

System.out.println("Processed payment via Square: $" + amt);

}

}

// File: PayPalAdapter.java

public class PayPalAdapter implements PaymentProcessor {

private final PayPalGateway paypal;

public PayPalAdapter(PayPalGateway paypal) {

this.paypal = paypal;

}

@Override

public void processPayment(double amount) {

paypal.makePayment(amount);

}

}

// File: StripeAdapter.java

public class StripeAdapter implements PaymentProcessor {

private final StripeGateway stripe;

public StripeAdapter(StripeGateway stripe) {

this.stripe = stripe;

}

@Override

public void processPayment(double amount) {

stripe.pay(amount);

}

}

// File: SquareAdapter.java

public class SquareAdapter implements PaymentProcessor {

private final SquareGateway square;

public SquareAdapter(SquareGateway square) {

this.square = square;

}

@Override

public void processPayment(double amount) {

square.doTransaction(amount);

}

}

// File: AdapterPatternTest.java

public class AdapterPatternTest {

public static void main(String[] args) {

PaymentProcessor paypalProcessor = new PayPalAdapter(new PayPalGateway());

paypalProcessor.processPayment(150.0);

PaymentProcessor stripeProcessor = new StripeAdapter(new StripeGateway());

stripeProcessor.processPayment(200.0);

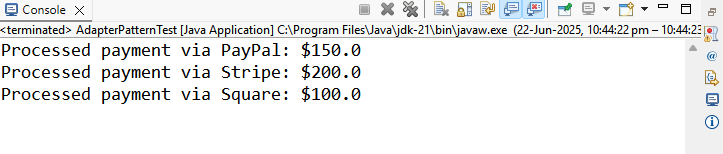
PaymentProcessor squareProcessor = new SquareAdapter(new SquareGateway());

squareProcessor.processPayment(100.0);

}

}

OUTPUT:



**Exercise 5: Implementing the Decorator Pattern**

// File: Notifier.java

public interface Notifier {

void send(String message);

}

// File: EmailNotifier.java

public class EmailNotifier implements Notifier {

@Override

public void send(String message) {

System.out.println("Email Notification: " + message);

}

}

// File: NotifierDecorator.java

public abstract class NotifierDecorator implements Notifier {

protected Notifier wrappee;

public NotifierDecorator(Notifier notifier) {

this.wrappee = notifier;

}

@Override

public void send(String message) {

wrappee.send(message);

}

}

// File: SMSNotifierDecorator.java

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

System.out.println("SMS Notification: " + message);

}

}

// File: SlackNotifierDecorator.java

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

System.out.println("Slack Notification: " + message);

}

}

// File: DecoratorPatternTest.java

public class DecoratorPatternTest {

public static void main(String[] args) {

Notifier notifier = new EmailNotifier();

// Add SMS and Slack as decorators

Notifier smsNotifier = new SMSNotifierDecorator(notifier);

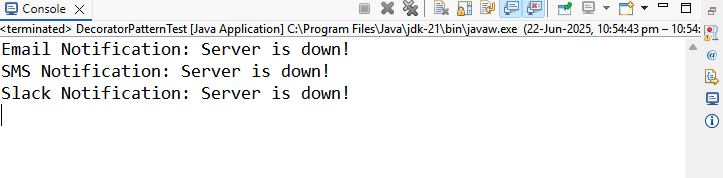
Notifier fullNotifier = new SlackNotifierDecorator(smsNotifier);

fullNotifier.send("Server is down!");

}

}

OUTPUT:



**Exercise 6: Implementing the Proxy Pattern**

// File: Image.java

public interface Image {

void display();

}

// File: RealImage.java

public class RealImage implements Image {

private final String filename;

public RealImage(String filename) {

this.filename = filename;

loadFromRemoteServer();

}

private void loadFromRemoteServer() {

System.out.println("Loading image from remote server: " + filename);

}

@Override

public void display() {

System.out.println("Displaying image: " + filename);

}

}

// File: ProxyImage.java

public class ProxyImage implements Image {

private final String filename;

private RealImage realImage;

public ProxyImage(String filename) {

this.filename = filename;

}

@Override

public void display() {

if (realImage == null) {

realImage = new RealImage(filename);

}

realImage.display();

}

}

// File: ProxyPatternTest.java

public class ProxyPatternTest {

public static void main(String[] args) {

Image image1 = new ProxyImage("design-pattern.png");

// First call - lazy loading

image1.display();

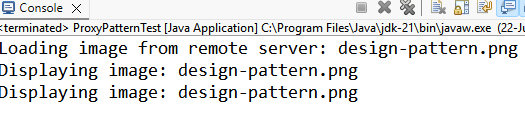
// Second call - cached

image1.display();

}

}

OUTPUT:



**Exercise 7: Implementing the Observer Pattern**

// File: Stock.java

public interface Stock {

void registerObserver(Observer observer);

void removeObserver(Observer observer);

void notifyObservers();

}

// File: Observer.java

public interface Observer {

void update(String stockName, double newPrice);

}

// File: StockMarket.java

import java.util.ArrayList;

import java.util.List;

public class StockMarket implements Stock {

private final List<Observer> observers = new ArrayList<>();

private String stockName;

private double stockPrice;

public void setStock(String stockName, double price) {

this.stockName = stockName;

this.stockPrice = price;

notifyObservers();

}

@Override

public void registerObserver(Observer observer) {

observers.add(observer);

}

@Override

public void removeObserver(Observer observer) {

observers.remove(observer);

}

@Override

public void notifyObservers() {

for (Observer observer : observers) {

observer.update(stockName, stockPrice);

}

}

}

// File: MobileApp.java

public class MobileApp implements Observer {

private final String appId;

public MobileApp(String appId) {

this.appId = appId;

}

@Override

public void update(String stockName, double newPrice) {

System.out.println("MobileApp [" + appId + "]: " + stockName + " updated to $" + newPrice);

}

}

// File: WebApp.java

public class WebApp implements Observer {

private final String user;

public WebApp(String user) {

this.user = user;

}

@Override

public void update(String stockName, double newPrice) {

System.out.println("WebApp [" + user + "]: " + stockName + " updated to $" + newPrice);

}

}

// File: ObserverPatternTest.java

public class ObserverPatternTest {

public static void main(String[] args) {

StockMarket market = new StockMarket();

Observer mobileClient = new MobileApp("M123");

Observer webClient = new WebApp("webUserA");

market.registerObserver(mobileClient);

market.registerObserver(webClient);

market.setStock("AAPL", 198.75);

market.setStock("GOOG", 2750.00);

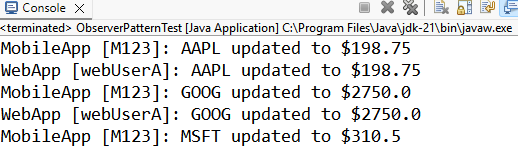
market.removeObserver(webClient);

market.setStock("MSFT", 310.50);

}

}

OUTPUT:



**Exercise 8: Implementing the Strategy Pattern**

// File: PaymentStrategy.java

public interface PaymentStrategy {

void pay(double amount);

}

// File: CreditCardPayment.java

public class CreditCardPayment implements PaymentStrategy {

private final String cardNumber;

public CreditCardPayment(String cardNumber) {

this.cardNumber = cardNumber;

}

@Override

public void pay(double amount) {

System.out.println("Paid $" + amount + " using Credit Card: " + cardNumber);

}

}

// File: PayPalPayment.java

public class PayPalPayment implements PaymentStrategy {

private final String email;

public PayPalPayment(String email) {

this.email = email;

}

@Override

public void pay(double amount) {

System.out.println("Paid $" + amount + " using PayPal account: " + email);

}

}

// File: PaymentContext.java

public class PaymentContext {

private PaymentStrategy paymentStrategy;

public void setPaymentStrategy(PaymentStrategy paymentStrategy) {

this.paymentStrategy = paymentStrategy;

}

public void executePayment(double amount) {

if (paymentStrategy == null) {

throw new IllegalStateException("Payment strategy not set.");

}

paymentStrategy.pay(amount);

}

}

// File: StrategyPatternTest.java

public class StrategyPatternTest {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

context.setPaymentStrategy(new CreditCardPayment("4111-XXXX-XXXX-1234"));

context.executePayment(250.00);

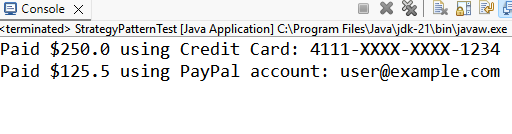
context.setPaymentStrategy(new PayPalPayment("user@example.com"));

context.executePayment(125.50);

}

}

OUTPUT:



**Exercise 9: Implementing the Command Pattern**

// File: Command.java

public interface Command {

void execute();

}

// File: Light.java

public class Light {

public void turnOn() {

System.out.println("Light is ON");

}

public void turnOff() {

System.out.println("Light is OFF");

}

}

// File: LightOnCommand.java

public class LightOnCommand implements Command {

private final Light light;

public LightOnCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOn();

}

}

// File: LightOffCommand.java

public class LightOffCommand implements Command {

private final Light light;

public LightOffCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOff();

}

}

// File: RemoteControl.java

public class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

if (command != null) {

command.execute();

} else {

System.out.println("No command set.");

}

}

}

// File: CommandPatternTest.java

public class CommandPatternTest {

public static void main(String[] args) {

Light livingRoomLight = new Light();

Command lightOn = new LightOnCommand(livingRoomLight);

Command lightOff = new LightOffCommand(livingRoomLight);

RemoteControl remote = new RemoteControl();

remote.setCommand(lightOn);

remote.pressButton();

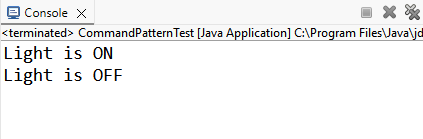
remote.setCommand(lightOff);

remote.pressButton();

}

}

OUTPUT:



**Exercise 10: Implementing the MVC Pattern**

// File: Student.java

public class Student {

private String name;

private String id;

private String grade;

public Student(String name, String id, String grade) {

this.name = name;

this.id = id;

this.grade = grade;

}

// Getters and setters

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getGrade() {

return grade;

}

public void setGrade(String grade) {

this.grade = grade;

}

}

// File: StudentView.java

public class StudentView {

public void displayStudentDetails(String name, String id, String grade) {

System.out.println("Student Details:");

System.out.println("Name : " + name);

System.out.println("ID : " + id);

System.out.println("Grade: " + grade);

}

}

// File: StudentController.java

public class StudentController {

private final Student model;

private final StudentView view;

public StudentController(Student model, StudentView view) {

this.model = model;

this.view = view;

}

// Model accessors via controller

public void setStudentName(String name) {

model.setName(name);

}

public String getStudentName() {

return model.getName();

}

public void setStudentId(String id) {

model.setId(id);

}

public String getStudentId() {

return model.getId();

}

public void setStudentGrade(String grade) {

model.setGrade(grade);

}

public String getStudentGrade() {

return model.getGrade();

}

public void updateView() {

view.displayStudentDetails(model.getName(), model.getId(), model.getGrade());

}

}

// File: MVCPatternTest.java

public class MVCPatternTest {

public static void main(String[] args) {

// Model

Student student = new Student("Alice", "S001", "A");

// View

StudentView view = new StudentView();

// Controller

StudentController controller = new StudentController(student, view);

// Initial display

controller.updateView();

// Modify data through controller

controller.setStudentName("Bob");

controller.setStudentGrade("B+");

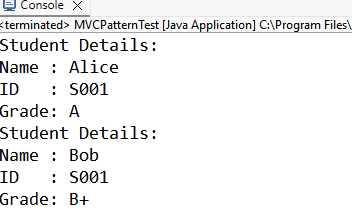
// Display updated view

controller.updateView();

}

}

OUTPUT:



**Exercise 11: Implementing Dependency Injection**

// File: CustomerRepository.java

public interface CustomerRepository {

String findCustomerById(String id);

}

// File: CustomerRepositoryImpl.java

public class CustomerRepositoryImpl implements CustomerRepository {

@Override

public String findCustomerById(String id) {

// Mock database logic

return "Customer[id=" + id + ", name=John Doe]";

}

}

// File: CustomerService.java

public class CustomerService {

private final CustomerRepository customerRepository;

// Constructor Injection

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public void displayCustomer(String id) {

String customer = customerRepository.findCustomerById(id);

System.out.println("Retrieved: " + customer);

}

}

// File: DependencyInjectionTest.java

public class DependencyInjectionTest {

public static void main(String[] args) {

CustomerRepository repository = new CustomerRepositoryImpl(); // Create dependency

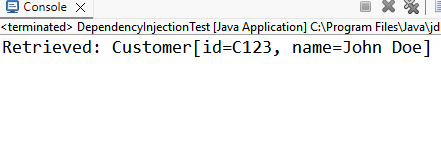
CustomerService service = new CustomerService(repository); // Inject via constructor

service.displayCustomer("C123");

}

}

OUTPUT:



ALGORITHMS AND DATA STRUCTURES

**Exercise 1: Inventory Management System**

// File: Product.java

public class Product {

private final String productId;

private String productName;

private int quantity;

private double price;

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

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

// Getters and setters

public String getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public int getQuantity() {

return quantity;

}

public double getPrice() {

return price;

}

public void setProductName(String productName) {

this.productName = productName;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public void setPrice(double price) {

this.price = price;

}

public String toString() {

return String.format("Product[%s] %s | Qty: %d | $%.2f", productId, productName, quantity, price);

}

}

// File: InventoryManager.java

import java.util.HashMap;

import java.util.Map;

public class InventoryManager {

private final Map<String, Product> inventory = new HashMap<>();

public void addProduct(Product product) {

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

System.out.println("Product already exists.");

} else {

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

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

}

}

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

Product product = inventory.get(productId);

if (product != null) {

product.setProductName(name);

product.setQuantity(quantity);

product.setPrice(price);

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

} else {

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

}

}

public void deleteProduct(String productId) {

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

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

} else {

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

}

}

public void printInventory() {

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

System.out.println(p);

}

}

}

// File: InventoryTest.java

public class InventoryTest {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

Product p1 = new Product("P001", "Keyboard", 50, 25.99);

Product p2 = new Product("P002", "Mouse", 75, 19.49);

Product p3 = new Product("P003", "Monitor", 30, 199.99);

manager.addProduct(p1);

manager.addProduct(p2);

manager.addProduct(p3);

manager.printInventory();

manager.updateProduct("P002", "Wireless Mouse", 70, 29.99);

manager.deleteProduct("P001");

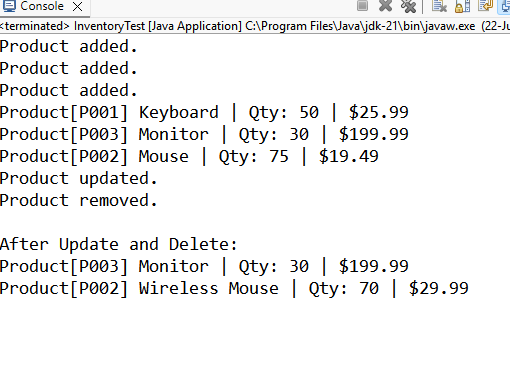
System.out.println("\nAfter Update and Delete:");

manager.printInventory();

}

}

OUTPUT:



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

// File: Product.java

public class Product {

private final String productId;

private final String productName;

private final String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public String getCategory() {

return category;

}

@Override

public String toString() {

return String.format("Product[%s] %s (%s)", productId, productName, category);

}

}

// File: ProductSearch.java

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

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

for (Product product : products) {

if (product.getProductName().equalsIgnoreCase(targetName)) {

return product;

}

}

return null;

}

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

int left = 0;

int right = sortedProducts.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

int cmp = sortedProducts[mid].getProductName().compareToIgnoreCase(targetName);

if (cmp == 0) {

return sortedProducts[mid];

} else if (cmp < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void sortByName(Product[] products) {

Arrays.sort(products, Comparator.comparing(Product::getProductName, String.CASE\_INSENSITIVE\_ORDER));

}

}

// File: SearchTest.java

public class SearchTest {

public static void main(String[] args) {

Product[] products = {

new Product("P001", "Laptop", "Electronics"),

new Product("P002", "Shoes", "Footwear"),

new Product("P003", "Phone", "Electronics"),

new Product("P004", "Bag", "Accessories"),

new Product("P005", "Watch", "Accessories")

};

System.out.println("=== Linear Search ===");

Product foundLinear = ProductSearch.linearSearch(products, "Phone");

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

System.out.println("\n=== Binary Search ===");

ProductSearch.sortByName(products);

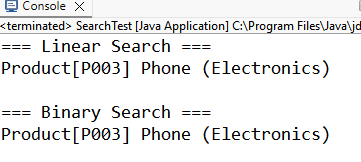
Product foundBinary = ProductSearch.binarySearch(products, "Phone");

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

}

}

OUTPUT:



**Exercise 3: Sorting Customer Orders**

// File: Order.java

public class Order {

private final String orderId;

private final String customerName;

private final double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public double getTotalPrice() {

return totalPrice;

}

public String toString() {

return String.format("Order[%s] %s - $%.2f", orderId, customerName, totalPrice);

}

}

// File: OrderSorter.java

public class OrderSorter {

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 - i - 1; j++) {

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

Order temp = orders[j];

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

orders[j + 1] = temp;

swapped = true;

}

}

if (!swapped) break;

}

}

public static void quickSort(Order[] orders) {

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

}

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

if (low < high) {

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

quickSort(orders, low, pivotIndex - 1);

quickSort(orders, pivotIndex + 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;

}

}

// File: SortTest.java

public class SortTest {

public static void main(String[] args) {

Order[] orders1 = {

new Order("O001", "Alice", 120.50),

new Order("O002", "Bob", 75.00),

new Order("O003", "Charlie", 250.75),

new Order("O004", "Dana", 50.25)

};

Order[] orders2 = orders1.clone(); // for Quick Sort

System.out.println("🔁 Bubble Sort:");

OrderSorter.bubbleSort(orders1);

for (Order o : orders1) System.out.println(o);

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

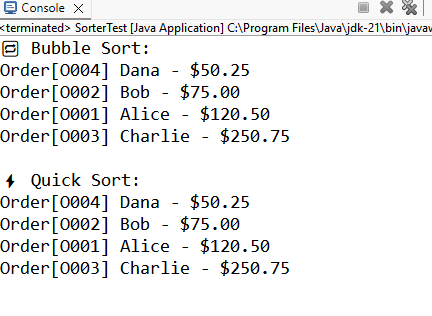
OrderSorter.quickSort(orders2);

for (Order o : orders2) System.out.println(o);

}

}

OUTPUT:



**Exercise 4: Employee Management System**

// File: Employee.java

public class Employee {

private final String employeeId;

private String name;

private String position;

private double salary;

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

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

public String getEmployeeId() {

return employeeId;

}

public String toString() {

return String.format("Employee[%s] %s - %s - $%.2f", employeeId, name, position, salary);

}

}

// File: EmployeeManager.java

public class EmployeeManager {

private final Employee[] employees;

private int size = 0;

public EmployeeManager(int capacity) {

employees = new Employee[capacity];

}

public boolean addEmployee(Employee emp) {

if (size >= employees.length) {

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

return false;

}

employees[size++] = emp;

return true;

}

public Employee searchEmployee(String empId) {

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

if (employees[i].getEmployeeId().equals(empId)) {

return employees[i];

}

}

return null;

}

public boolean deleteEmployee(String empId) {

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

if (employees[i].getEmployeeId().equals(empId)) {

// Shift elements left

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

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

}

employees[--size] = null;

return true;

}

}

return false;

}

public void displayAll() {

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

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

}

}

}

// File: EmployeeSystemTest.java

public class EmployeeSystemTest {

public static void main(String[] args) {

EmployeeManager manager = new EmployeeManager(5);

manager.addEmployee(new Employee("E001", "Alice", "Manager", 85000));

manager.addEmployee(new Employee("E002", "Bob", "Developer", 70000));

manager.addEmployee(new Employee("E003", "Charlie", "Designer", 65000));

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

manager.displayAll();

System.out.println("\nSearching E002:");

System.out.println(manager.searchEmployee("E002"));

System.out.println("\nDeleting E002:");

manager.deleteEmployee("E002");

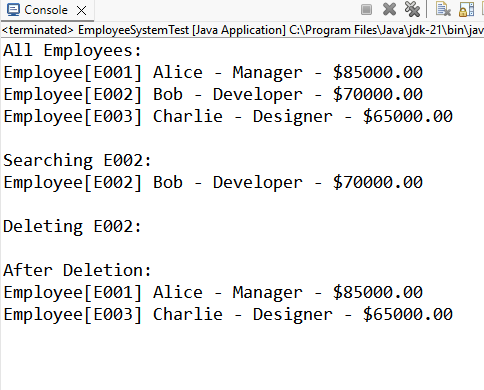
System.out.println("\nAfter Deletion:");

manager.displayAll();

}

}

OUTPUT:



**Exercise 5: Task Management System**

// File: Task.java

public class Task {

private final String taskId;

private final String taskName;

private String status;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

public String getTaskId() {

return taskId;

}

public void setStatus(String status) {

this.status = status;

}

public String toString() {

return String.format("Task[%s] %s - %s", taskId, taskName, status);

}

}

// File: TaskLinkedList.java

public class TaskLinkedList {

private static class Node {

Task task;

Node next;

Node(Task task) {

this.task = task;

this.next = null;

}

}

private Node head;

public void addTask(Task task) {

Node newNode = new Node(task);

if (head == null) {

head = newNode;

return;

}

Node curr = head;

while (curr.next != null) {

curr = curr.next;

}

curr.next = newNode;

}

public Task searchTask(String taskId) {

Node curr = head;

while (curr != null) {

if (curr.task.getTaskId().equals(taskId)) {

return curr.task;

}

curr = curr.next;

}

return null;

}

public boolean deleteTask(String taskId) {

if (head == null) return false;

if (head.task.getTaskId().equals(taskId)) {

head = head.next;

return true;

}

Node prev = head;

Node curr = head.next;

while (curr != null) {

if (curr.task.getTaskId().equals(taskId)) {

prev.next = curr.next;

return true;

}

prev = curr;

curr = curr.next;

}

return false;

}

public void displayTasks() {

Node curr = head;

while (curr != null) {

System.out.println(curr.task);

curr = curr.next;

}

}

}

// File: TaskManagerTest.java

public class TaskManagerTest {

public static void main(String[] args) {

TaskLinkedList taskList = new TaskLinkedList();

taskList.addTask(new Task("T001", "Design UI", "Pending"));

taskList.addTask(new Task("T002", "Write Backend", "In Progress"));

taskList.addTask(new Task("T003", "Testing", "Pending"));

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

taskList.displayTasks();

System.out.println("\nSearch Task T002:");

Task found = taskList.searchTask("T002");

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

System.out.println("\nDelete Task T001:");

boolean deleted = taskList.deleteTask("T001");

System.out.println(deleted ? "Deleted successfully." : "Delete failed.");

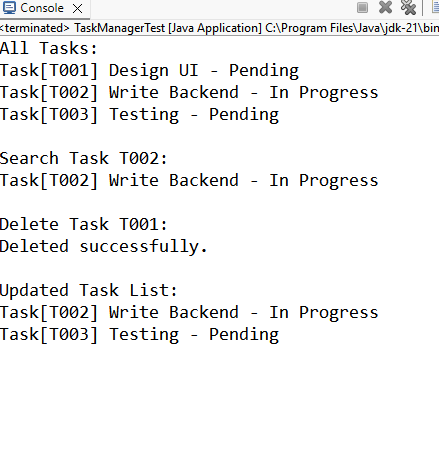
System.out.println("\nUpdated Task List:");

taskList.displayTasks();

}

}

OUTPUT:



**Exercise 6: Library Management System**

// File: Book.java

public class Book {

private final String bookId;

private final String title;

private final String author;

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

this.bookId = bookId;

this.title = title;

this.author = author;

}

public String getTitle() {

return title;

}

public String toString() {

return String.format("Book[%s] \"%s\" by %s", bookId, title, author);

}

}

// File: BookSearch.java

import java.util.Arrays;

import java.util.Comparator;

public class BookSearch {

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

for (Book book : books) {

if (book.getTitle().equalsIgnoreCase(targetTitle)) {

return book;

}

}

return null;

}

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

int left = 0;

int right = books.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

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

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

else right = mid - 1;

}

return null;

}

public static void sortByTitle(Book[] books) {

Arrays.sort(books, Comparator.comparing(Book::getTitle, String.CASE\_INSENSITIVE\_ORDER));

}

}

// File: LibraryTest.java

public class LibraryTest {

public static void main(String[] args) {

Book[] books = {

new Book("B001", "The Alchemist", "Paulo Coelho"),

new Book("B002", "1984", "George Orwell"),

new Book("B003", "Brave New World", "Aldous Huxley"),

new Book("B004", "Moby Dick", "Herman Melville")

};

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

Book result1 = BookSearch.linearSearch(books, "1984");

System.out.println(result1 != null ? result1 : "Not found");

BookSearch.sortByTitle(books); // Required for binary search

System.out.println("\n⚡ Binary Search:");

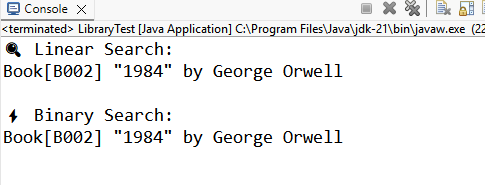
Book result2 = BookSearch.binarySearch(books, "1984");

System.out.println(result2 != null ? result2 : "Not found");

}

}

OUTPUT:



**Exercise 7: Financial Forecasting**

// File: FinancialForecast.java

public class FinancialForecast {

// Recursive method to calculate future value

public static double futureValue(double initialAmount, double rate, int years) {

if (years == 0) return initialAmount;

return futureValue(initialAmount, rate, years - 1) \* (1 + rate);

}

// Optimized version using tail recursion with memoization

public static double futureValueMemo(double[] cache, double rate, int years) {

if (cache[years] != 0) return cache[years];

cache[years] = futureValueMemo(cache, rate, years - 1) \* (1 + rate);

return cache[years];

}

}

// File: ForecastTest.java

public class ForecastTest {

public static void main(String[] args) {

double initial = 1000;

double rate = 0.05; // 5% annual growth

int years = 10;

System.out.println("📈 Recursive Forecast:");

double fv = FinancialForecast.futureValue(initial, rate, years);

System.out.printf("Future Value after %d years: $%.2f%n", years, fv);

System.out.println("\n⚡ Optimized Recursive Forecast:");

double[] memo = new double[years + 1];

memo[0] = initial;

double optimized = FinancialForecast.futureValueMemo(memo, rate, years);

System.out.printf("Future Value after %d years: $%.2f%n", years, optimized);

}

}

OUTPUT:

