**Digital Nurture 4.0 Java FSE**

**Week 1 – Design Patterns and Principles, Data Structure and Algorithm**

**(Additional)**

**Design Patterns and Principles:**

**Exercise 03 – Implementing the Builder Pattern**

**Code:**

public class computer {  
  
 public static void main(String[] args) {  
  
 computer gamingRig = new computer.Builder()  
 .setCPU("Intel Core i9")  
 .setRAM("32GB")  
 .setStorage("1TB SSD")  
 .setGPU("NVIDIA RTX 4080")  
 .enableBluetooth(true)  
 .enableWiFi(true)  
 .build();  
  
 computer officePC = new computer.Builder()  
 .setCPU("Intel Core i5")  
 .setRAM("8GB")  
 .setStorage("512GB HDD")  
 .enableBluetooth(false)  
 .enableWiFi(true)  
 .build();  
  
 System.*out*.println("Gaming Rig:\n" + gamingRig);  
 System.*out*.println("Office PC:\n" + officePC);  
 }  
  
 private final String CPU;  
 private final String RAM;  
 private final String storage;  
 private final String GPU;  
 private final boolean isBluetoothEnabled;  
 private final boolean isWiFiEnabled;  
  
 private computer(Builder builder) {  
 this.CPU = builder.CPU;  
 this.RAM = builder.RAM;  
 this.storage = builder.storage;  
 this.GPU = builder.GPU;  
 this.isBluetoothEnabled = builder.isBluetoothEnabled;  
 this.isWiFiEnabled = builder.isWiFiEnabled;  
 }  
  
 @Override  
 public String toString() {  
 return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", storage=" + storage +  
 ", GPU=" + GPU + ", Bluetooth=" + isBluetoothEnabled +  
 ", WiFi=" + isWiFiEnabled + "]";  
 }  
  
 public static class Builder {  
 private String CPU;  
 private String RAM;  
 private String storage;  
 private String GPU;  
 private boolean isBluetoothEnabled;  
 private boolean isWiFiEnabled;  
  
 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 enableBluetooth(boolean value) {  
 this.isBluetoothEnabled = value;  
 return this;  
 }  
  
 public Builder enableWiFi(boolean value) {  
 this.isWiFiEnabled = value;  
 return this;  
 }  
  
 public computer build() {  
 return new computer(this);  
 }  
 }  
}

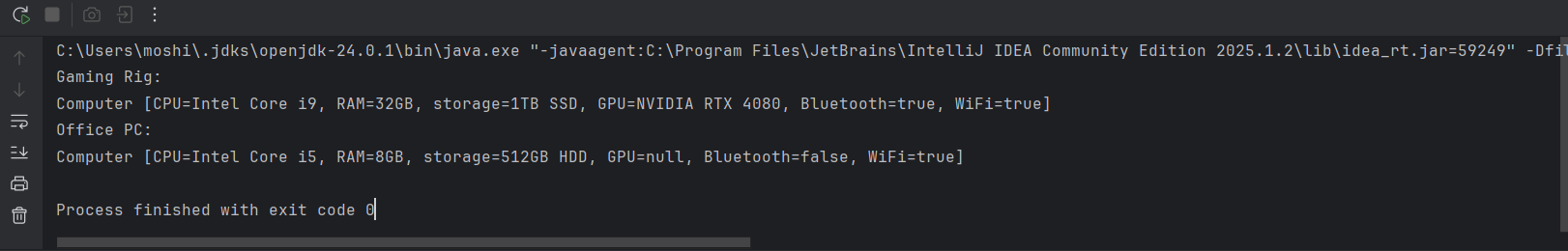
**Output:**

**Gaming Rig:**

Computer [CPU=Intel Core i9, RAM=32GB, storage=1TB SSD, GPU=NVIDIA RTX 4080, Bluetooth=true, WiFi=true]

**Office PC:**

Computer [CPU=Intel Core i5, RAM=8GB, storage=512GB HDD, GPU=null, Bluetooth=false, WiFi=true]



**Exercise 04 – Implementing the Adapter Pattern**

**Code:**

public class PaymentProcessor {

public static void main(String[] args) {

PaymentProcessorInterface payPalProcessor = new PayPalAdapter(new PayPalGateway());

payPalProcessor.processPayment(250.0);

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

stripeProcessor.processPayment(400.0);

}

}

interface PaymentProcessorInterface {

void processPayment(double amount);

}

class PayPalGateway {

public void sendPayment(double amount) {

System.out.println("PayPal processed payment of ₹" + amount);

}

}

class StripeGateway {

public void makeStripePayment(double total) {

System.out.println("Stripe processed payment of ₹" + total);

}

}

class PayPalAdapter implements PaymentProcessorInterface {

private PayPalGateway gateway;

public PayPalAdapter(PayPalGateway gateway) {

this.gateway = gateway;

}

@Override

public void processPayment(double amount) {

gateway.sendPayment(amount);

}

}

class StripeAdapter implements PaymentProcessorInterface {

private StripeGateway gateway;

public StripeAdapter(StripeGateway gateway) {

this.gateway = gateway;

}

@Override

public void processPayment(double amount) {

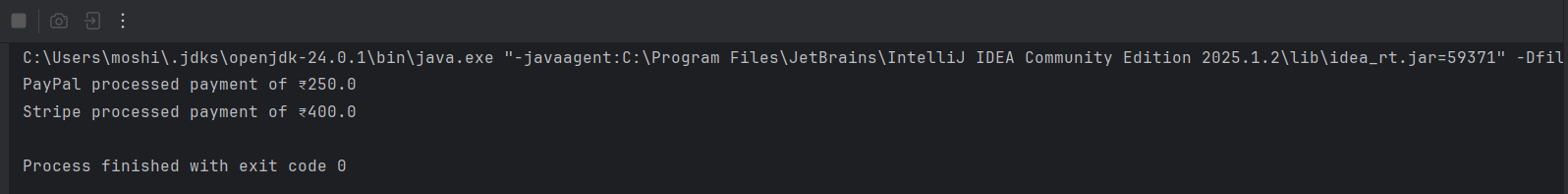
gateway.makeStripePayment(amount);

}

}

**Output:**

PayPal processed payment of ₹250.0

****Stripe processed payment of ₹400.0

**Exercise 05 – Implementing the Decorator Pattern**

**Code:**

public class Notifier {

public static void main(String[] args) {

NotifierInterface notifier = new SlackNotifierDecorator(

new SMSNotifierDecorator(

new EmailNotifier()));

notifier.send("System update completed successfully.");

}

}

interface NotifierInterface

void send(String message);

}

class EmailNotifier implements NotifierInterface {

public void send(String message) {

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

}

}

abstract class NotifierDecorator implements NotifierInterface {

protected NotifierInterface wrapped;

public NotifierDecorator(NotifierInterface notifier) {

this.wrapped = notifier;

}

}

class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(NotifierInterface notifier) {

super(notifier);

}

public void send(String message) {

wrapped.send(message);

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

}

}

class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(NotifierInterface notifier) {

super(notifier);

}

public void send(String message) {

wrapped.send(message);

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

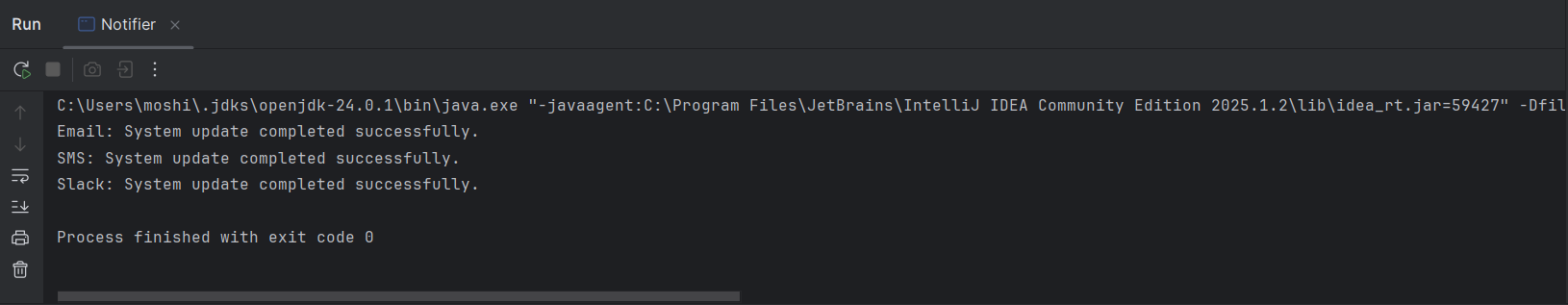
}

}

**Output:**

Email: System update completed successfully.

SMS: System update completed successfully.

Slack: System update completed successfully.

**Exercise 06 – Implementing the Proxy Pattern**

**Code:**

public class Image {

public static void main(String[] args) {

ImageInterface image1 = new ProxyImage("cat.jpg");

ImageInterface image2 = new ProxyImage("dog.jpg");

image1.display(); // Loads and displays

System.out.println();

image1.display(); // Uses cache

System.out.println();

image2.display(); // Loads and displays

}

}

interface ImageInterface {

void display();

}

class RealImage implements ImageInterface {

private String filename;

public RealImage(String filename) {

this.filename = filename;

loadFromRemoteServer();

}

private void loadFromRemoteServer() {

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

}

public void display() {

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

}

}

class ProxyImage implements ImageInterface {

private RealImage realImage;

private String filename;

public ProxyImage(String filename) {

this.filename = filename;

}

public void display() {

if (realImage == null) {

realImage = new RealImage(filename);

} else {

System.out.println(filename + " is already loaded. Using cached version.");

}

realImage.display();

}

}

**Output:**

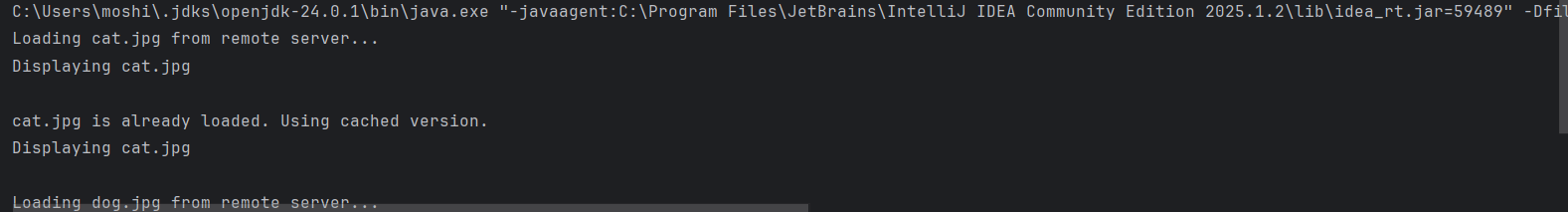
Loading cat.jpg from remote server...

Displaying cat.jpg

cat.jpg is already loaded. Using cached version.

Displaying cat.jpg

Loading dog.jpg from remote server...

Displaying dog.jpg

**Exercise 07 – Implementing the Observer Pattern**

**Code:**

public class Stock {

public static void main(String[] args) {

StockMarket stockMarket = new StockMarket();

Observer mobile = new MobileApp("Moshika's Mobile");

Observer web = new WebApp("Moshika's Web");

stockMarket.registerObserver(mobile);

stockMarket.registerObserver(web);

stockMarket.setPrice("TSLA", 752.60);

System.out.println();

stockMarket.setPrice("GOOGL", 1398.25);

}

}

interface StockInterface {

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObservers(String stock, double price);

}

interface Observer {

void update(String stock, double price);

}

class StockMarket implements StockInterface {

private java.util.List<Observer> observers = new java.util.ArrayList<>();

public void registerObserver(Observer o) {

observers.add(o);

}

public void removeObserver(Observer o) {

observers.remove(o);

}

public void notifyObservers(String stock, double price) {

for (Observer o : observers) {

o.update(stock, price);

}

}

public void setPrice(String stock, double price) {

System.out.println("Stock Update: " + stock + " = ₹" + price);

notifyObservers(stock, price);

}

}

class MobileApp implements Observer {

private String name;

public MobileApp(String name) {

this.name = name;

}

public void update(String stock, double price) {

System.out.println("[" + name + "] Mobile Notification → " + stock + " at ₹" + price);

}

}

class WebApp implements Observer {

private String name;

public WebApp(String name) {

this.name = name;

}

public void update(String stock, double price) {

System.out.println("[" + name + "] Web Notification → " + stock + " at ₹" + price);

}

}

**Output:**

Stock Update: TSLA = ₹752.6

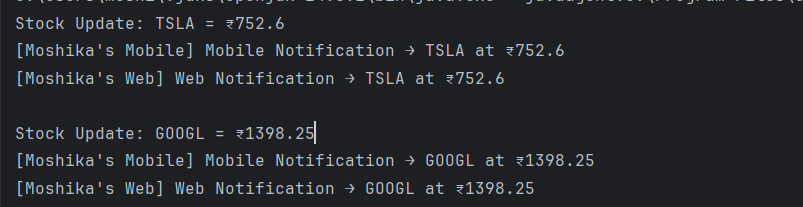
[Moshika's Mobile] Mobile Notification → TSLA at ₹752.6

[Moshika's Web] Web Notification → TSLA at ₹752.6

Stock Update: GOOGL = ₹1398.25

[Moshika's Mobile] Mobile Notification → GOOGL at ₹1398.25

[Moshika's Web] Web Notification → GOOGL at ₹1398.25

****

**Exercise 08 – Implementing the Strategy Pattern**

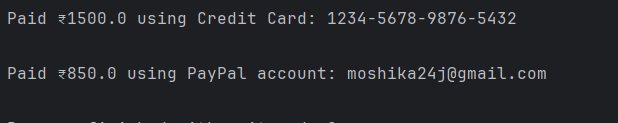
**Code:**

public class PaymentStrategy {  
  
 public static void main(String[] args) {  
 PaymentContext context;  
  
 context = new PaymentContext(new CreditCardPayment("1234-5678-9876-5432", "Moshika", "789"));  
 context.pay(1500.0);  
  
 System.*out*.println();  
  
 context = new PaymentContext(new PayPalPayment("moshika24j@gmail.com", "secure123"));  
 context.pay(850.0);  
 }  
}  
  
interface PaymentMethod {  
 void pay(double amount);  
}  
  
class CreditCardPayment implements PaymentMethod {  
 private String cardNumber;  
 private String cardHolder;  
 private String cvv;  
  
 public CreditCardPayment(String cardNumber, String cardHolder, String cvv) {  
 this.cardNumber = cardNumber;  
 this.cardHolder = cardHolder;  
 this.cvv = cvv;  
 }  
  
 public void pay(double amount) {  
 System.*out*.println("Paid ₹" + amount + " using Credit Card: " + cardNumber);  
 }  
}  
  
class PayPalPayment implements PaymentMethod {  
 private String email;  
 private String password;  
  
 public PayPalPayment(String email, String password) {  
 this.email = email;  
 this.password = password;  
 }  
  
 public void pay(double amount) {  
 System.*out*.println("Paid ₹" + amount + " using PayPal account: " + email);  
 }  
}  
  
class PaymentContext {  
 private PaymentMethod strategy;  
  
 public PaymentContext(PaymentMethod strategy) {  
 this.strategy = strategy;  
 }  
  
 public void pay(double amount) {  
 strategy.pay(amount);  
 }  
}

**Output:**

Paid ₹1500.0 using Credit Card: 1234-5678-9876-5432

Paid ₹850.0 using PayPal account: [moshika24j@gmail.com](mailto:moshika24j@gmail.com)



**Exercise 09 – Implementing the Command Pattern**

**Code:**

public class CommandDemo {

public static void main(String[] args) {

Light livingRoomLight = new Light("Living Room");

Command onCommand = new LightOnCommand(livingRoomLight);

Command offCommand = new LightOffCommand(livingRoomLight);

RemoteControl remote = new RemoteControl();

remote.setCommand(onCommand);

remote.pressButton();

remote.setCommand(offCommand);

remote.pressButton();

}

}

interface Command {

void execute();

}

class Light {

private String location;

public Light(String location) {

this.location = location;

}

public void turnOn() {

System.out.println(location + " light is ON");

}

public void turnOff() {

System.out.println(location + " light is OFF");

}

}

class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOn();

}

}

class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOff();

}

}

class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

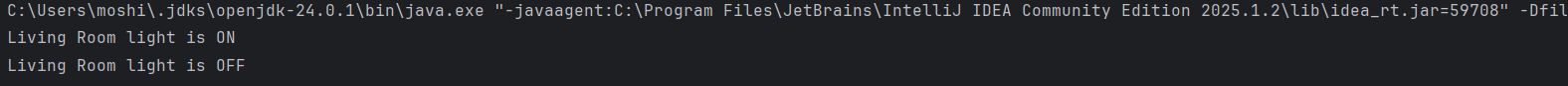
command.execute();

}

}

**Output:**

Living Room light is ON

Living Room light is OFF

**Exercise 10 – Implementing the MVC Pattern**

**Code:**

public class Student {

public static void main(String[] args) {

StudentModel model = new StudentModel();

model.setId("S1234");

model.setName("Moshika");

model.setGrade("A");

StudentView view = new StudentView();

StudentController controller = new StudentController(model, view);

controller.updateView();

System.out.println();

controller.setStudentName("Moshika R");

controller.setStudentGrade("A+");

controller.updateView();

}

}

class StudentModel {

private String id;

private String name;

private String grade;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getGrade() {

return grade;

}

public void setGrade(String grade) {

this.grade = grade;

}

}

class StudentView {

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

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

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

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

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

}

}

class StudentController {

private StudentModel model;

private StudentView view;

public StudentController(StudentModel model, StudentView view) {

this.model = model;

this.view = view;

}

public void setStudentName(String name) {

model.setName(name);

}

public void setStudentGrade(String grade) {

model.setGrade(grade);

}

public void updateView() {

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

}

}

**Output:**

Student Details:

ID : S1234

Name : Moshika

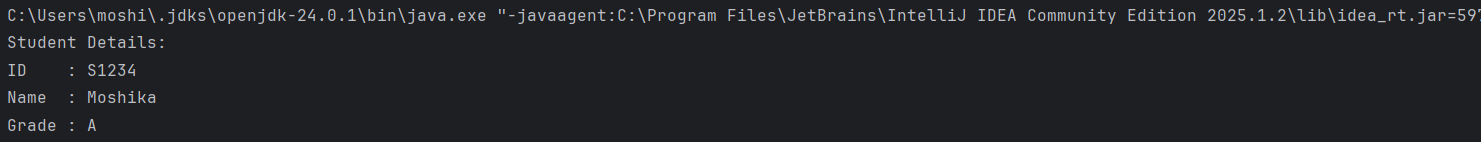
Grade : A

Student Details:

ID : S1234

Name : Moshika R

Grade : A+



**Exercise 11 – Implementing the Dependency Injection**

**Code:**

public class CustomerService {

public static void main(String[] args) {

CustomerRepository repository = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repository);

Customer customer = service.findCustomer("C102");

if (customer != null) {

System.out.println("Customer Found: " + customer.getName() + ", Grade: " + customer.getTier());

} else {

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

}

}

private CustomerRepository repository;

public CustomerService(CustomerRepository repository) {

this.repository = repository;

}

public Customer findCustomer(String id) {

return repository.findCustomerById(id);

}

}

interface CustomerRepository {

Customer findCustomerById(String id);

}

class CustomerRepositoryImpl implements CustomerRepository {

public Customer findCustomerById(String id) {

if (id.equals("C102")) {

return new Customer("C102", "Moshika", "Gold");

}

return null;

}

}

class Customer {

private String id;

private String name;

private String tier;

public Customer(String id, String name, String tier) {

this.id = id;

this.name = name;

this.tier = tier;

}

public String getId() {

return id;

}

public String getName() **{**

return name;

}

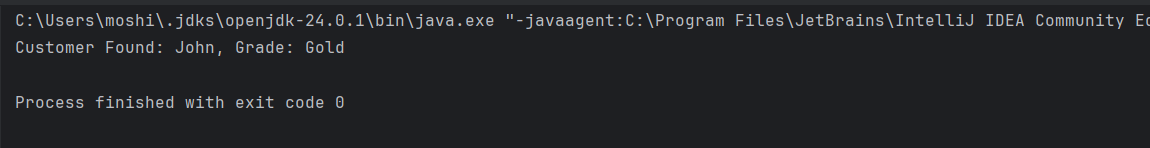
public String getTier() {

return tier;

}

}

**Output:**

Customer Found: John, Grade: Gold

**Data Structures and Algorithm:**

**Exercise 01 – Inventory Management System**

**Code:**

import java.util.HashMap;

import java.util.Map;

public class Inventory {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

manager.addProduct(new Product("P201", "Wireless Mouse", 50, 999.0));

manager.addProduct(new Product("P202", "Mechanical Keyboard", 30, 3499.0));

manager.addProduct(new Product("P203", "External HDD 1TB", 20, 4199.0));

manager.addProduct(new Product("P204", "USB-C Charger", 60, 1499.0));

System.out.println("\nInitial Inventory:");

manager.printInventory();

manager.updateProduct("P202", 25, 3299.0);

manager.deleteProduct("P203");

System.out.println("\nUpdated Inventory:");

manager.printInventory();

}

}

class Product {

private 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;

}

public void update(int quantity, double price) {

this.quantity = quantity;

this.price = price;

}

public String getId() {

return productId;

}

public String toString() {

return productId + " | " + productName + " | " + quantity + " units | ₹" + price;

}

}

class InventoryManager {

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

public void addProduct(Product product) {

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

}

public void updateProduct(String productId, int newQuantity, double newPrice) {

Product product = inventory.get(productId);

if (product != null) {

product.update(newQuantity, newPrice);

}

}

public void deleteProduct(String productId) {

inventory.remove(productId);

}

public void printInventory() {

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

System.out.println(product);

}

}

}

**Output:**

Initial Inventory:

P203 | External HDD 1TB | 20 units | ₹4199.0

P202 | Mechanical Keyboard | 30 units | ₹3499.0

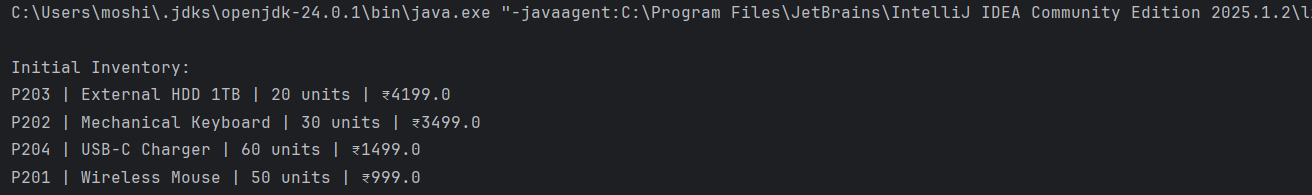
P204 | USB-C Charger | 60 units | ₹1499.0

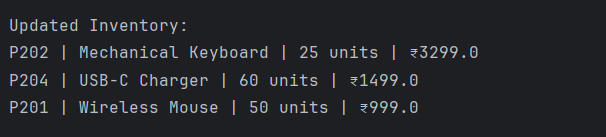
P201 | Wireless Mouse | 50 units | ₹999.0

Updated Inventory:

P202 | Mechanical Keyboard | 25 units | ₹3299.0

P204 | USB-C Charger | 60 units | ₹1499.0

P201 | Wireless Mouse | 50 units | ₹999.0



**Exercise 03 – Sorting Customer Orders**

**Code:**

class Order {

String orderId;

String customerName;

double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public String toString() {

return orderId + " | " + customerName + " | ₹" + totalPrice;

}

}

public class CustomerOrderSorter {

public static void main(String[] args) {

Order[] orders = {

new Order("O101", "Moshika", 4999.0),

new Order("O102", "Kavya", 2599.0),

new Order("O103", "Latha", 7899.0),

new Order("O104", "Nikhil", 1499.0)

};

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

printOrders(orders);

Order[] bubbleSorted = orders.clone();

bubbleSort(bubbleSorted);

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

printOrders(bubbleSorted);

Order[] quickSorted = orders.clone();

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

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

printOrders(quickSorted);

}

static void bubbleSort(Order[] arr) {

for (int i = 0; i < arr.length - 1; i++) {

for (int j = 0; j < arr.length - 1 - i; j++) {

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

Order temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

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

if (low < high) {

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

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

double pivot = arr[high].totalPrice;

int i = low - 1;

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

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

i++;

Order temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

Order temp = arr[i + 1];

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

arr[high] = temp;

return i + 1;

}

static void printOrders(Order[] arr) {

for (Order o : arr) {

System.out.println(o);

}

}

}

**Output:**

**Original Orders:**

O101 | Moshika | ₹4999.0

O102 | Kavya | ₹2599.0

O103 | Latha | ₹7899.0

O104 | Nikhil | ₹1499.0

**Bubble Sorted Orders:**

O104 | Nikhil | ₹1499.0

O102 | Kavya | ₹2599.0

O101 | Moshika | ₹4999.0

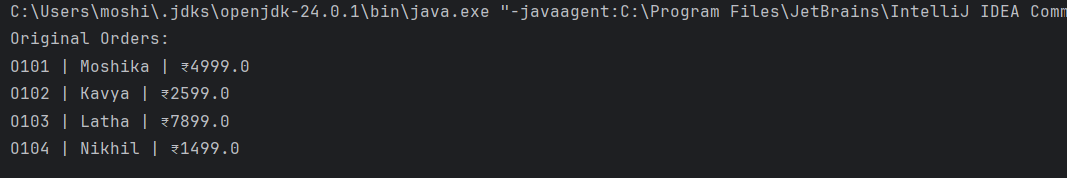
O103 | Latha | ₹7899.0

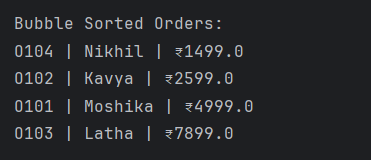
**Quick Sorted Orders:**

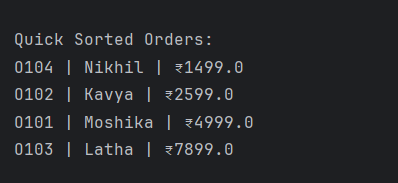
O104 | Nikhil | ₹1499.0

O102 | Kavya | ₹2599.0

O101 | Moshika | ₹4999.0

O103 | Latha | ₹7899.0





**Exercise 04 – Employee Management System**

**Code:**

import java.util.Scanner;

public class EmployeeSystem {

public static void main(String[] args) {

EmployeeManager manager = new EmployeeManager(10); // capacity 10

manager.addEmployee(new Employee(101, "Moshika", "Developer", 80000));

manager.addEmployee(new Employee(102, "Kavya", "HR", 60000));

manager.addEmployee(new Employee(103, "Ravi", "Manager", 95000));

manager.addEmployee(new Employee(104, "Nikhil", "Analyst", 72000));

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

manager.traverse();

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

manager.search(102);

System.out.println("\nDeleting Employee ID 103:");

manager.delete(103);

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

manager.traverse();

}

}

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 String toString() {

return employeeId + " | " + name + " | " + position + " | ₹" + salary;

}

}

class EmployeeManager {

private Employee[] employees;

private int count = 0;

public EmployeeManager(int size) {

employees = new Employee[size];

}

public void addEmployee(Employee e) {

if (count >= employees.length) {

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

return;

}

employees[count++] = e;

}

public void search(int empId) {

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

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

System.out.println("Found: " + employees[i]);

return;

}

}

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

}

public void traverse() {

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

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

}

}

public void delete(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 ID " + empId + " deleted.");

return;

}

}

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

}

}

**Output:**

All Employees:

101 | Moshika | Developer | ₹80000.0

102 | Kavya | HR | ₹60000.0

103 | Ravi | Manager | ₹95000.0

104 | Nikhil | Analyst | ₹72000.0

Searching for Employee ID 102:

Found: 102 | Kavya | HR | ₹60000.0

Deleting Employee ID 103:

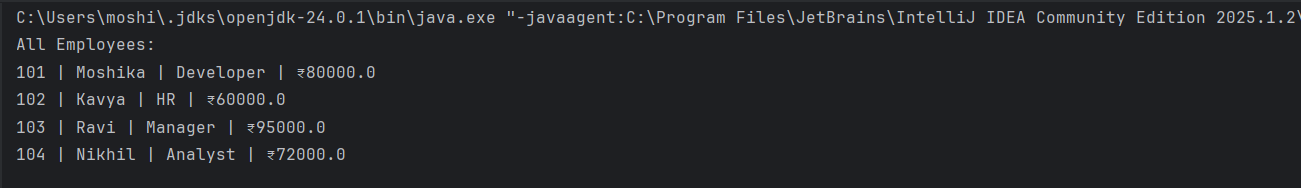
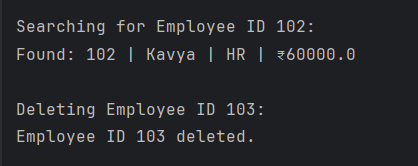
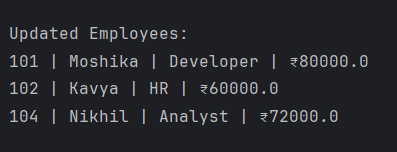
Employee ID 103 deleted.

Updated Employees:

101 | Moshika | Developer | ₹80000.0

102 | Kavya | HR | ₹60000.0

104 | Nikhil | Analyst | ₹72000.0



**Exercise 05 – Task Management System**

**Code:**

public class TaskManager {

public static void main(String[] args) {

TaskList list = new TaskList();

list.addTask(new Task(1, "Fix login bug", "Pending"));

list.addTask(new Task(2, "Update dashboard UI", "In Progress"));

list.addTask(new Task(3, "Deploy to staging", "Pending"));

list.addTask(new Task(4, "Write release notes", "Completed"));

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

list.traverse();

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

list.search(2);

System.out.println("\nDelete Task ID 3:");

list.delete(3);

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

list.traverse();

}

}

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;

}

public String toString() {

return taskId + " | " + taskName + " | " + status;

}

}

class TaskNode {

Task data;

TaskNode next;

public TaskNode(Task data) {

this.data = data;

this.next = null;

}

}

class TaskList {

private TaskNode head;

public void addTask(Task task) {

TaskNode newNode = new TaskNode(task);

if (head == null) {

head = newNode;

} else {

TaskNode current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

}

public void traverse() {

TaskNode current = head;

while (current != null) {

System.out.println(current.data);

current = current.next;

}

}

public void search(int taskId) {

TaskNode current = head;

while (current != null) {

if (current.data.taskId == taskId) {

System.out.println("Found: " + current.data);

return;

}

current = current.next;

}

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

}

public void delete(int taskId) {

if (head == null) return;

if (head.data.taskId == taskId) {

head = head.next;

return;

}

TaskNode current = head;

while (current.next != null) {

if (current.next.data.taskId == taskId) {

current.next = current.next.next;

return;

}

current = current.next;

}

}

}

**Output:**

All Tasks:

1 | Fix login bug | Pending

2 | Update dashboard UI | In Progress

3 | Deploy to staging | Pending

4 | Write release notes | Completed

Search Task ID 2:

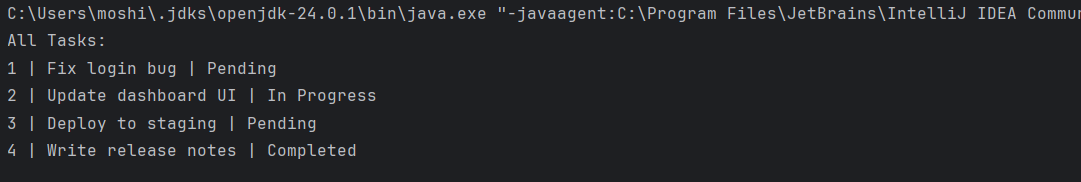
Found: 2 | Update dashboard UI | In Progress

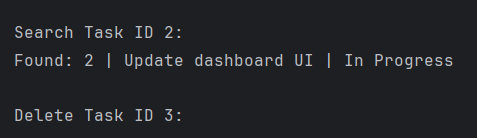
Delete Task ID 3:

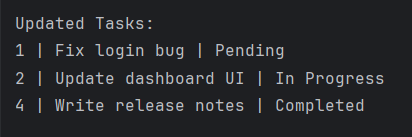
Updated Tasks:

1 | Fix login bug | Pending

2 | Update dashboard UI | In Progress

4 | Write release notes | Completed





**Exercise 06 – Library Management System**

**Code:**

import java.util.Arrays;

import java.util.Comparator;

public class Library {

public static void main(String[] args) {

Book[] books = {

new Book(101, "The Alchemist", "Paulo Coelho"),

new Book(102, "To Kill a Mockingbird", "Harper Lee"),

new Book(103, "Atomic Habits", "James Clear"),

new Book(104, "Brave New World", "Aldous Huxley"),

new Book(105, "1984", "George Orwell")

};

System.out.println("Linear Search for 'Atomic Habits':");

linearSearch(books, "Atomic Habits");

System.out.println("\nBinary Search for 'Brave New World':");

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

binarySearch(books, "Brave New World");

}

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

for (Book b : books) {

if (b.title.equalsIgnoreCase(title)) {

System.out.println("Found: " + b);

return;

}

}

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

}

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

int low = 0, high = books.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (cmp == 0) {

System.out.println("Found: " + books[mid]);

return;

} else if (cmp < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

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

}

}

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 String toString() {

return bookId + " | " + title + " | " + author;

}

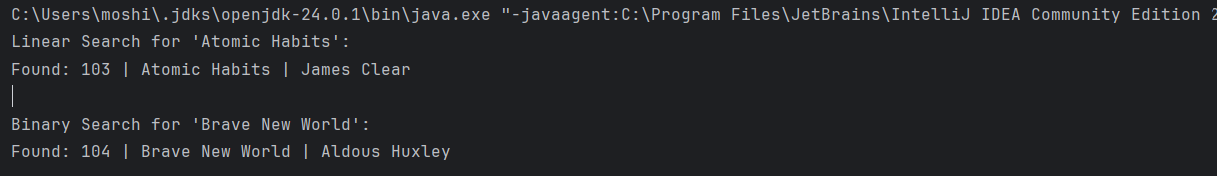
}

**Output:**

Linear Search for 'Atomic Habits':

Found: 103 | Atomic Habits | James Clear

Binary Search for 'Brave New World':

Found: 104 | Brave New World | Aldous Huxley