**WEEK - 1**

**Design principles & Patterns**

1) Exercise 1: Implementing the Singleton Pattern

CODE:

public class ex1 {

private static final ex1 instance = new ex1();

private ex1() {

System.out.println("instance created");

}

public static ex1 getInstance() {

return instance;

}

public void Message() {

System.out.println("Hello from instance!");

}

public static void main(String[] args) {

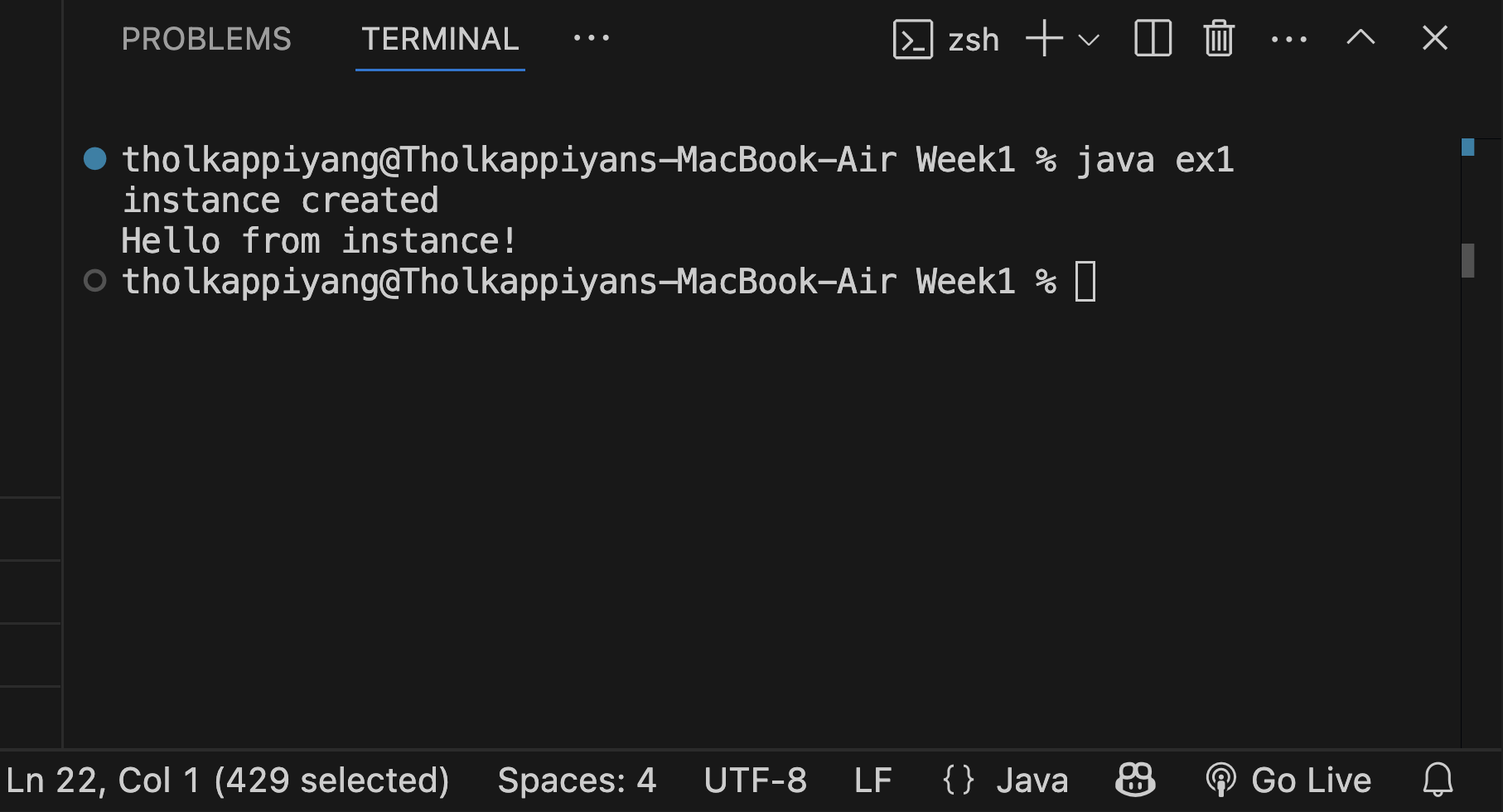
ex1 singleton = ex1.getInstance();

singleton.Message();

}

}

OUTPUT:



2) Exercise 2: Implementing the Factory Method Pattern

CODE:

// FactoryMethodPattern

interface Product {

void use();

}

class ConcreteProductA implements Product {

public void use() {

System.out.println("Using Product A");

}

}

class ConcreteProductB implements Product {

public void use() {

System.out.println("Using Product B");

}

}

abstract class Creator {

public abstract Product createProduct();

}

class ConcreteCreatorA extends Creator {

public Product createProduct() {

return new ConcreteProductA();

}

}

class ConcreteCreatorB extends Creator {

public Product createProduct() {

return new ConcreteProductB();

}

}

public class ex2 {

public static void main(String[] args) {

Creator creatorA = new ConcreteCreatorA();

Product productA = creatorA.createProduct();

productA.use();

Creator creatorB = new ConcreteCreatorB();

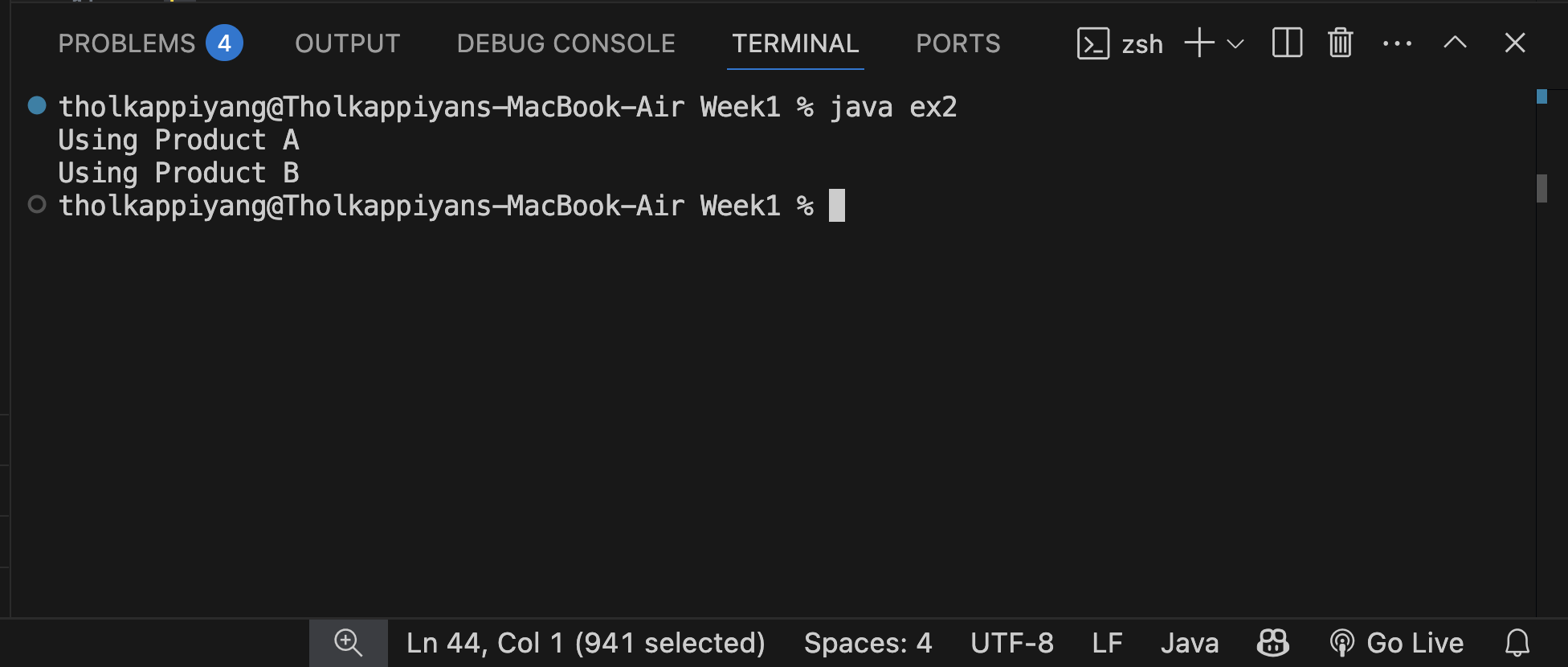
Product productB = creatorB.createProduct();

productB.use();

}

}

OUTPUT:



3) Exercise 3: E-commerce Platform Search Function

CODE:

import java.util.\*;

class Product {

private String id;

private String name;

private String category;

private double price;

public Product(String id, String name, String category, double price) {

this.id = id;

this.name = name.toLowerCase();

this.category = category.toLowerCase();

this.price = price;

}

public Product() {

}

public String getName() { return name; }

public String getCategory() { return category; }

public double getPrice() { return price; }

public void display() {

System.out.println(id + " | " + name + " | " + category + " | $" + price);

}

}

interface SearchStrategy {

List<Product> search(List<Product> products, String keyword);

}

class KeywordSearchStrategy implements SearchStrategy {

public List<Product> search(List<Product> products, String keyword) {

List<Product> result = new ArrayList<>();

String key = keyword.toLowerCase();

for (Product p : products) {

if (p.getName().contains(key) || p.getCategory().contains(key)) {

result.add(p);

}

}

return result;

}

}

class DisplayService {

public void show(List<Product> products) {

if (products.isEmpty()) {

System.out.println("No products found.");

} else {

for (Product p : products) {

p.display();

}

}

}

}

class ECommercePlatform {

private List<Product> products = new ArrayList<>();

private SearchStrategy searchStrategy;

private DisplayService displayService;

public ECommercePlatform(SearchStrategy searchStrategy, DisplayService displayService) {

this.searchStrategy = searchStrategy;

this.displayService = displayService;

}

public void addProduct(Product p) {

products.add(p);

}

public void search(String keyword) {

List<Product> results = searchStrategy.search(products, keyword);

displayService.show(results);

}

}

public class ex3 {

public static void main(String[] args) {

SearchStrategy strategy = new KeywordSearchStrategy();

DisplayService display = new DisplayService();

ECommercePlatform platform = new ECommercePlatform(strategy, display);

platform.addProduct(new Product("P101", "iPhone 14", "Smartphones", 799.00));

platform.addProduct(new Product("P102", "Samsung Galaxy", "Smartphones", 699.00));

platform.addProduct(new Product("P103", "HP Pavilion", "Laptops", 649.00));

platform.addProduct(new Product("P104", "Dell Inspiron", "Laptops", 599.00));

platform.addProduct(new Product("P105", "Sony Headphones", "Accessories", 99.00));

Scanner sc = new Scanner(System.in);

System.out.print("Search for a product: ");

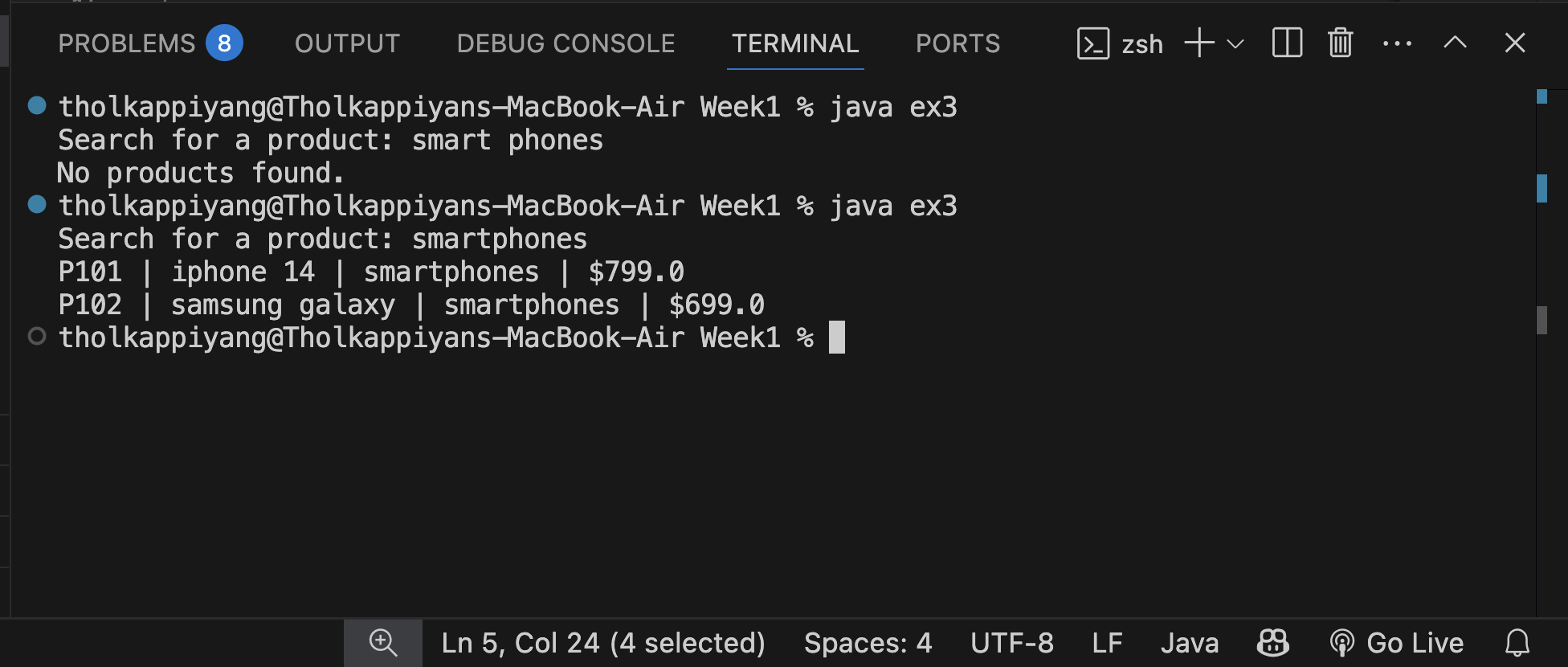
String input = sc.nextLine();

platform.search(input);

}

}

OUTPUT:



4) Exercise 7: Financial Forecasting

CODE:

// FinancialForecasting.java

import java.util.\*;

class RevenueInputService {

public double[] collectPastRevenues(Scanner scanner, int months) {

double[] revenues = new double[months];

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

System.out.print("Enter revenue for month " + (i + 1) + ": ₹");

revenues[i] = scanner.nextDouble();

}

return revenues;

}

}

class GrowthRateCalculator {

public double calculateAverageGrowthRate(double[] revenues) {

double totalGrowthRate = 0;

int growthPoints = 0;

for (int i = 1; i < revenues.length; i++) {

if (revenues[i - 1] != 0) {

double growth = (revenues[i] - revenues[i - 1]) / revenues[i - 1];

totalGrowthRate += growth;

growthPoints++;

}

}

return (growthPoints > 0) ? totalGrowthRate / growthPoints : 0;

}

}

class ForecastService {

public List<Double> forecast(double lastRevenue, int forecastMonths, double averageGrowthRate) {

List<Double> forecasts = new ArrayList<>();

double revenue = lastRevenue;

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

revenue \*= (1 + averageGrowthRate);

forecasts.add(revenue);

}

return forecasts;

}

}

class DisplayService {

public void showForecast(List<Double> forecasts, int startMonth) {

System.out.println("\n--- Forecast for Next " + forecasts.size() + " Months ---");

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

System.out.printf("Month %d Forecast: ₹%.2f\n", startMonth + i + 1, forecasts.get(i));

}

}

public void showGrowthRate(double rate) {

System.out.printf("\nAverage Monthly Growth Rate: %.2f%%\n", rate \* 100);

}

}

public class ex4 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of past months of data: ");

int months = scanner.nextInt();

RevenueInputService inputService = new RevenueInputService();

double[] revenues = inputService.collectPastRevenues(scanner, months);

GrowthRateCalculator calculator = new GrowthRateCalculator();

double averageGrowthRate = calculator.calculateAverageGrowthRate(revenues);

System.out.print("Enter number of months to forecast: ");

int forecastMonths = scanner.nextInt();

ForecastService forecaster = new ForecastService();

List<Double> forecasts = forecaster.forecast(revenues[months - 1], forecastMonths, averageGrowthRate);

DisplayService display = new DisplayService();

display.showForecast(forecasts, months);

display.showGrowthRate(averageGrowthRate);

scanner.close();

}

}

OUTPUT:

