Exercise 1: Singleton pattern

public class Singleton {

private static volatile Singleton instance;

private Singleton() {

if (instance != null) {

throw new RuntimeException("Use getInstance() method to get the single instance of this class.");

}

 }

public static Singleton getInstance() {

 if (instance == null) {

 synchronized (Singleton.class) {

 if (instance == null) {

 instance = new Singleton();

 }

 }

}

return instance;

public String getDescription() {

return "I am the Singleton instance!";

}

public static void main(String[] args){

       Singleton singleton1 = Singleton.getInstance();

       Singleton singleton2 = Singleton.getInstance();

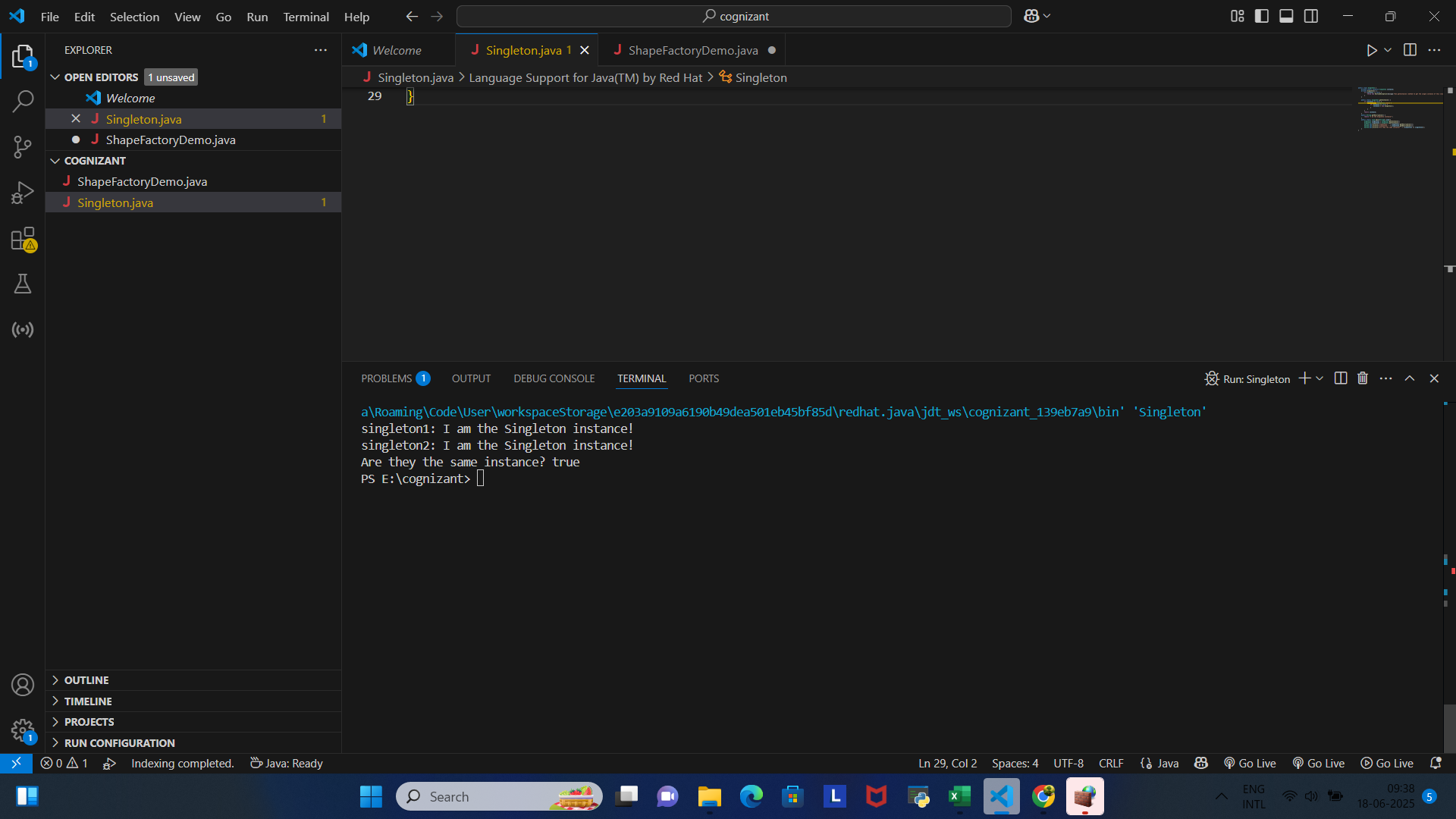
       System.out.println("singleton1: " + singleton1.getDescription());

       System.out.println("singleton2: " + singleton2.getDescription());

       System.out.println("Are they the same instance? " + (singleton1 == singleton2));

   }

}



Exercise 2:Factory Method pattern

interface Shape {

    void draw();

}

class Circle implements Shape {

    public void draw() {

        System.out.println("Drawing a Circle");

    }

}

class Rectangle implements Shape {

    public void draw() {

        System.out.println("Drawing a Rectangle");

    }

}

class ShapeFactory {

    public Shape getShape(String shapeType) {

        if (shapeType == null) {

            return null;

        }

        if (shapeType.equalsIgnoreCase("CIRCLE")) {

            return new Circle();

        } else if (shapeType.equalsIgnoreCase("RECTANGLE")) {

            return new Rectangle();

        }

        return null;

    }

}

public class ShapeFactoryDemo {

    public static void main(String[] args) {

        ShapeFactory shapeFactory = new ShapeFactory();

        Shape circle = shapeFactory.getShape("CIRCLE");

        if (circle != null) {

            circle.draw();

        }

        Shape rectangle = shapeFactory.getShape("RECTANGLE");

        if (rectangle != null) {

            rectangle.draw();

        }

        Shape invalid = shapeFactory.getShape("SQUARE");

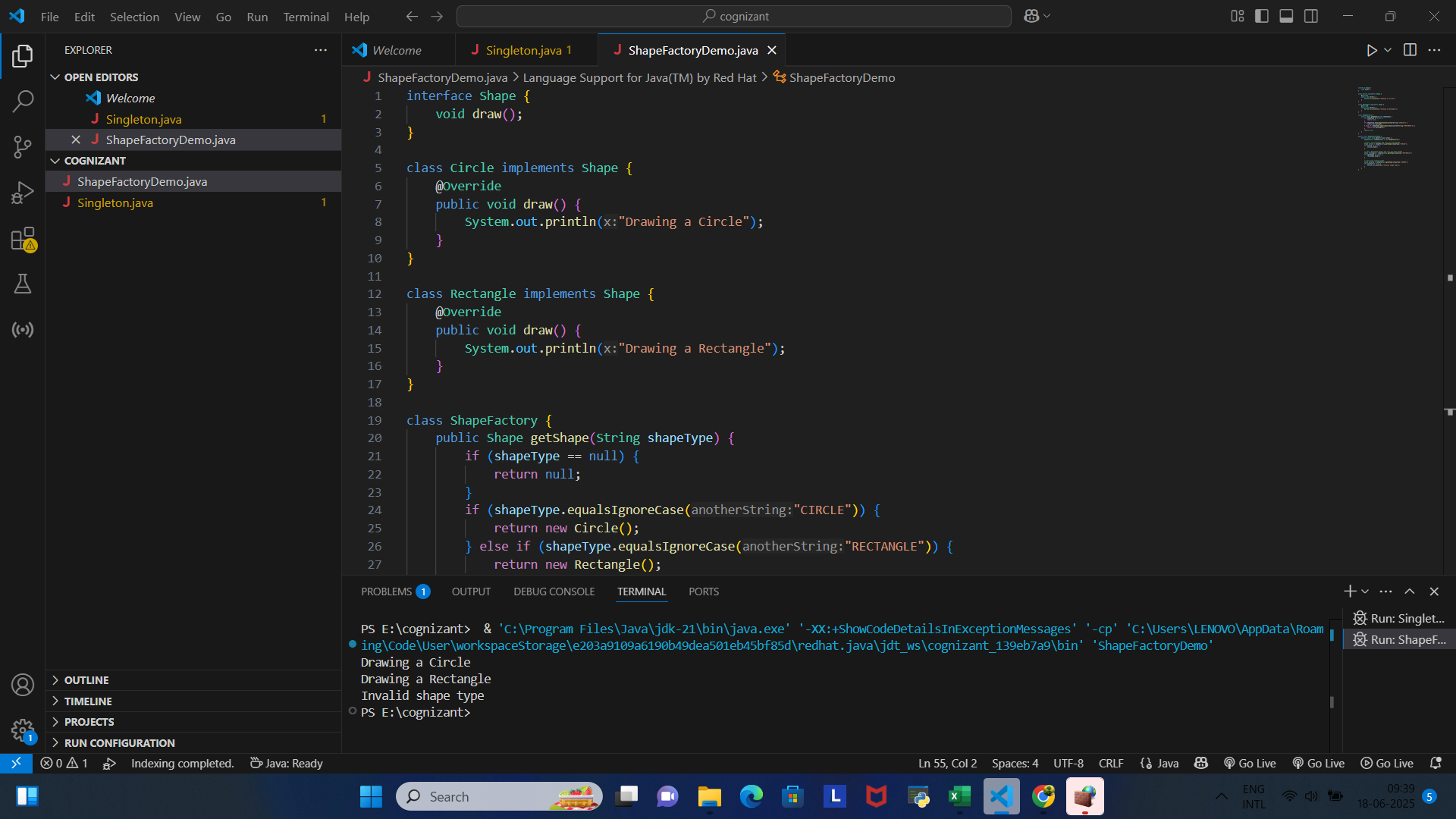
        if (invalid == null) {

            System.out.println("Invalid shape type");

        }

    }

}



Exercise 3:E-commerce platform search function

import java.util.\*;

class Product {

    int id;

    String name;

    String description;

    public Product(int id, String name, String description) {

        this.id = id;

        this.name = name;

        this.description = description;

    }

    public String toString() {

        return "Product{id=" + id + ", name='" + name + "', description='" + description + "'}";

    }

}

class TrieNode {

    Map<Character, TrieNode> children;

    List<Integer> productIds;

    public TrieNode() {

        children = new HashMap<>();

        productIds = new ArrayList<>();

    }

}

class EcommerceSearch {

    private Map<String, List<Integer>> invertedIndex;

    private TrieNode trieRoot;

    private List<Product> products;

    public EcommerceSearch() {

        invertedIndex = new HashMap<>();

        trieRoot = new TrieNode();

        products = new ArrayList<>();

    }

    public void addProduct(Product product) {

        products.add(product);

        indexProduct(product);

    }

    private void indexProduct(Product product) {

        String[] keywords = (product.name + " " + product.description).toLowerCase().split("\\W+");

        for (String keyword : keywords) {

            if (!keyword.isEmpty()) {

                invertedIndex.computeIfAbsent(keyword, k -> new ArrayList<>()).add(product.id);

                insertIntoTrie(keyword, product.id);

            }

        }

    }

    private void insertIntoTrie(String keyword, int productId) {

        TrieNode current = trieRoot;

        for (char c : keyword.toCharArray()) {

            current.children.computeIfAbsent(c, k -> new TrieNode());

            current = current.children.get(c);

        }

        current.productIds.add(productId);

    }

    public List<Product> search(String query) {

        if (query == null || query.trim().isEmpty()) {

            return new ArrayList<>();

        }

        String[] keywords = query.toLowerCase().split("\\W+");

        Map<Integer, Integer> productScore = new HashMap<>();

        for (String keyword : keywords) {

            List<Integer> productIds = invertedIndex.getOrDefault(keyword, new ArrayList<>());

            for (int id : productIds) {

                productScore.merge(id, 1, Integer::sum);

            }

        }

        List<Map.Entry<Integer, Integer>> sortedProducts = new ArrayList<>(productScore.entrySet());

        sortedProducts.sort((a, b) -> b.getValue().compareTo(a.getValue()));

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

        for (Map.Entry<Integer, Integer> entry : sortedProducts) {

            for (Product product : products) {

                if (product.id == entry.getKey()) {

                    results.add(product);

                    break;

                }

            }

        }

        return results;

    }

    public List<String> autocomplete(String prefix) {

        TrieNode current = trieRoot;

        for (char c : prefix.toLowerCase().toCharArray()) {

            if (!current.children.containsKey(c)) {

                return new ArrayList<>();

            }

            current = current.children.get(c);

        }

        Set<Integer> productIds = new HashSet<>();

        collectProductIds(current, productIds);

        List<String> suggestions = new ArrayList<>();

        for (Product product : products) {

            if (productIds.contains(product.id)) {

                suggestions.add(product.name);

            }

        }

        return suggestions;

    }

    private void collectProductIds(TrieNode node, Set<Integer> productIds) {

        productIds.addAll(node.productIds);

        for (TrieNode child : node.children.values()) {

            collectProductIds(child, productIds);

        }

    }

}

public class EcommerceSearchDemo {

    public static void main(String[] args) {

        EcommerceSearch searchEngine = new EcommerceSearch();

        searchEngine.addProduct(new Product(1, "Red Summer Dress", "Floral pattern, cotton, size M"));

        searchEngine.addProduct(new Product(2, "Blue Jacket", "Waterproof, winter, size L"));

        searchEngine.addProduct(new Product(3, "Red Hat", "Cotton, summer accessory"));

        System.out.println("Search results for 'red summer':");

        List<Product> results = searchEngine.search("red summer");

        for (Product product : results) {

            System.out.println(product);

        }

        System.out.println("\nAutocomplete suggestions for 're':");

        List<String> suggestions = searchEngine.autocomplete("re");

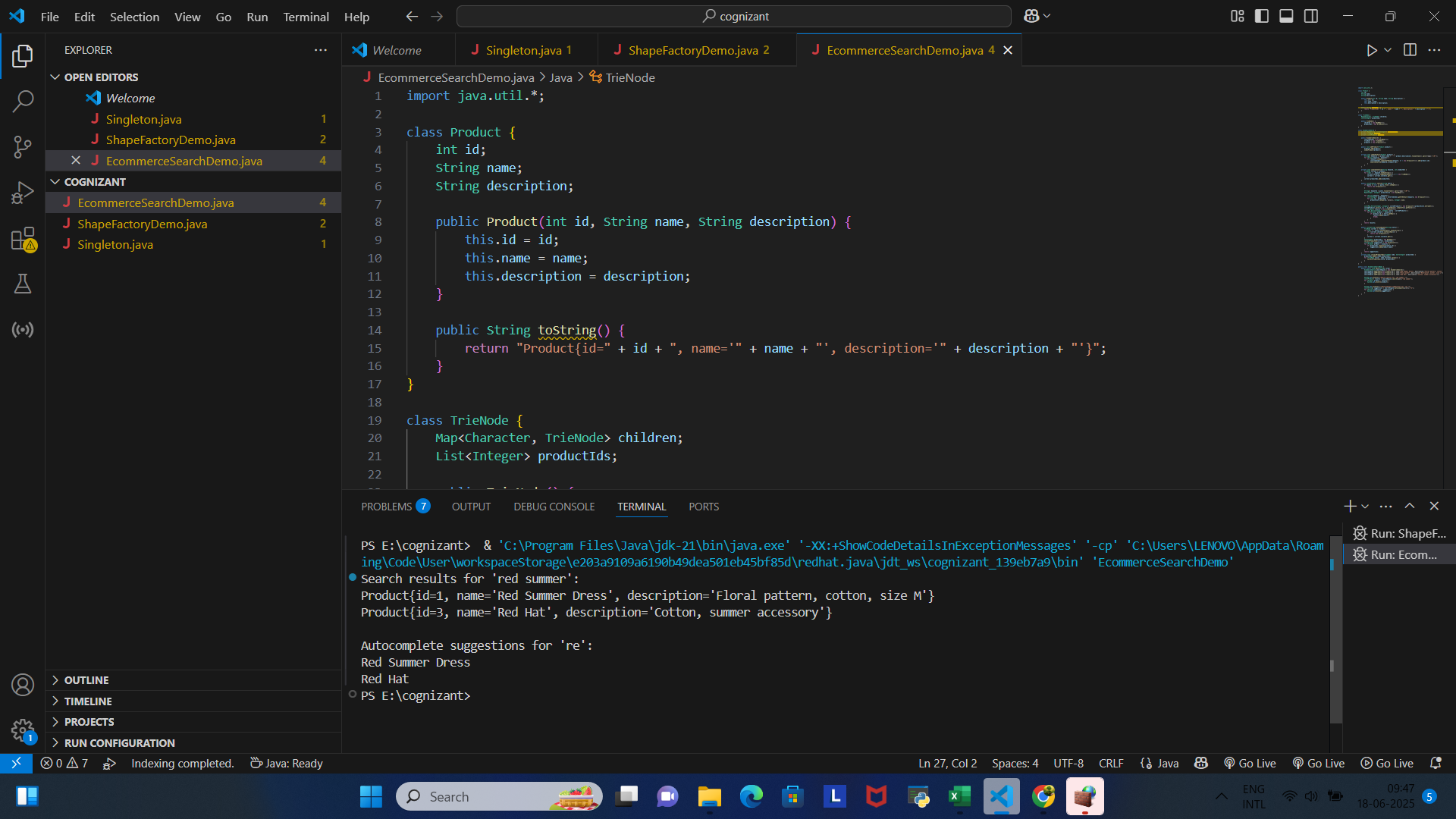
        for (String suggestion : suggestions) {

            System.out.println(suggestion);

        }

    }

}



Exercise 4: Financial Forecasting

import java.util.\*;

class Stock {

    String symbol;

    List<Double> historicalPrices;

    double forecastedPrice;

    double predictedReturn;

    public Stock(String symbol) {

        this.symbol = symbol;

        this.historicalPrices = new ArrayList<>();

        this.forecastedPrice = 0.0;

        this.predictedReturn = 0.0;

    }

    public void addPrice(double price) {

        historicalPrices.add(price);

    }

    public String toString() {

        return "Stock{symbol='" + symbol + "', forecastedPrice=" + String.format("%.2f", forecastedPrice) +

               ", predictedReturn=" + String.format("%.2f", predictedReturn \* 100) + "%}";

    }

}

class FinancialForecasting {

    private Map<String, Stock> stockData;

    private int windowSize;

    public FinancialForecasting(int windowSize) {

        this.stockData = new HashMap<>();

        this.windowSize = windowSize;

    }

    public void addStockPrice(String symbol, double price) {

        stockData.computeIfAbsent(symbol, k -> new Stock(symbol)).addPrice(price);

    }

    public void forecastPrices() {

        for (Stock stock : stockData.values()) {

            List<Double> prices = stock.historicalPrices;

            if (prices.size() >= windowSize) {

                double sum = 0.0;

                for (int i = prices.size() - windowSize; i < prices.size(); i++) {

                    sum += prices.get(i);

                }

                stock.forecastedPrice = sum / windowSize;

                double lastPrice = prices.get(prices.size() - 1);

                stock.predictedReturn = (stock.forecastedPrice - lastPrice) / lastPrice;

            }

        }

    }

    public List<Stock> getTopStocks(int n) {

        PriorityQueue<Stock> maxHeap = new PriorityQueue<>(

            (a, b) -> Double.compare(b.predictedReturn, a.predictedReturn)

        );

        for (Stock stock : stockData.values()) {

            if (stock.forecastedPrice > 0.0) {

                maxHeap.offer(stock);

            }

        }

        List<Stock> topStocks = new ArrayList<>();

        while (!maxHeap.isEmpty() && topStocks.size() < n) {

            topStocks.add(maxHeap.poll());

        }

        return topStocks;

    }

}

public class FinancialForecastingDemo {

    public static void main(String[] args) {

        FinancialForecasting forecaster = new FinancialForecasting(3);

        forecaster.addStockPrice("AAPL", 150.0);

        forecaster.addStockPrice("AAPL", 152.0);

        forecaster.addStockPrice("AAPL", 154.0);

        forecaster.addStockPrice("GOOGL", 2800.0);

        forecaster.addStockPrice("GOOGL", 2820.0);

        forecaster.addStockPrice("GOOGL", 2850.0);

        forecaster.addStockPrice("MSFT", 300.0);

        forecaster.addStockPrice("MSFT", 295.0);

        forecaster.addStockPrice("MSFT", 290.0);

        forecaster.forecastPrices();

        System.out.println("Top 2 stocks by predicted return:");

        List<Stock> topStocks = forecaster.getTopStocks(2);

        for (Stock stock : topStocks) {

            System.out.println(stock);

        }

    }

}

