**Exercise 1: Implementing the Singleton Pattern**

**Coding :**

public enum Singleton {

INSTANCE;

public void showMessage() {

System.out.println("Hello from Enum Singleton");

}

}

public class Main {

public static void main(String[] args) {

Singleton s1 = Singleton.INSTANCE;

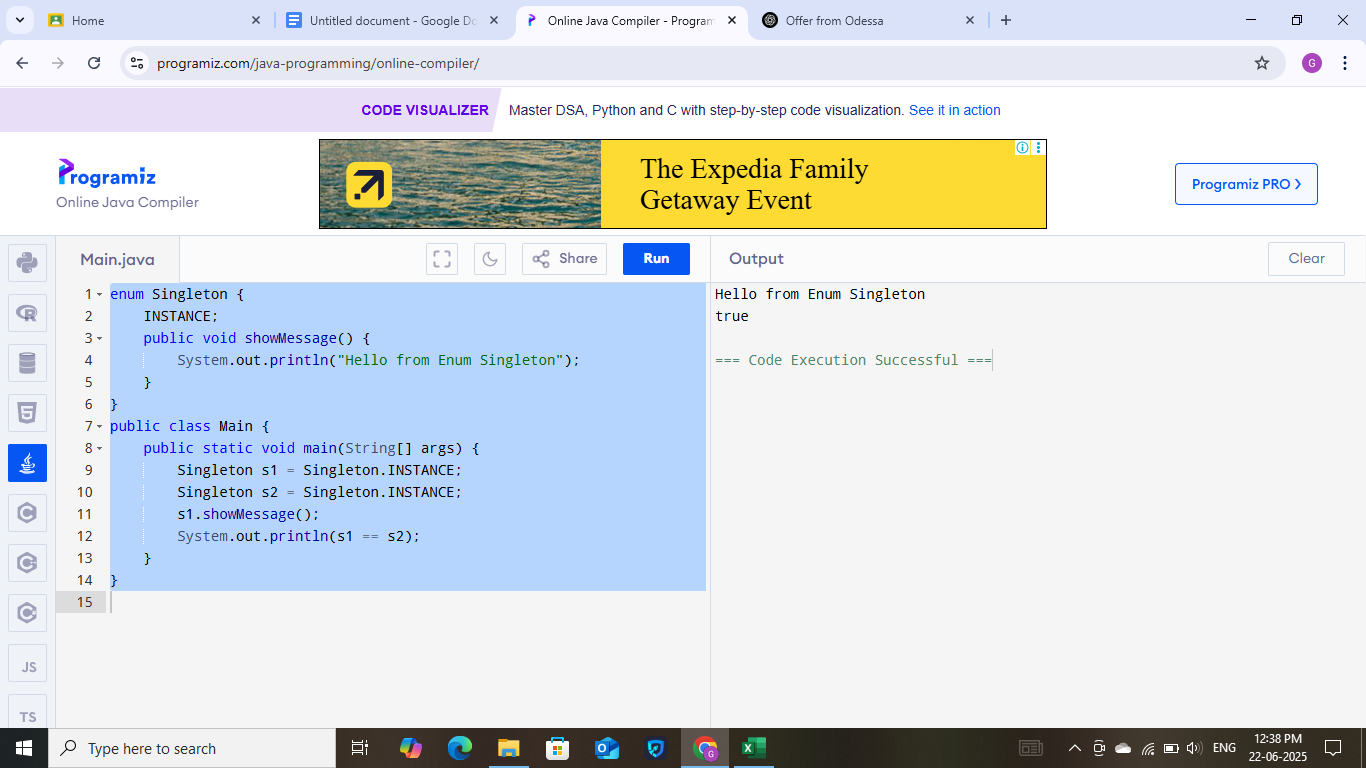
Singleton s2 = Singleton.INSTANCE;

s1.showMessage();

System.out.println(s1 == s2);

}

}



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

**Coding :**

import java.util.Map;

import java.util.function.Supplier;

interface Product {

void display();

}

class Book implements Product {

public void display() {

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

}

}

class Electronics implements Product {

public void display() {

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

}

}

class ProductFactory {

private static final Map<String, Supplier<Product>> MAP = Map.of(

"Book", Book::new,

"Electronics", Electronics::new

);

public static Product createProduct(String type) {

Supplier<Product> product = MAP.get(type);

if (product != null) return product.get();

throw new IllegalArgumentException("Unknown type: " + type);

}

}

public class Main {

public static void main(String[] args) {

Product p1 = ProductFactory.createProduct("Book");

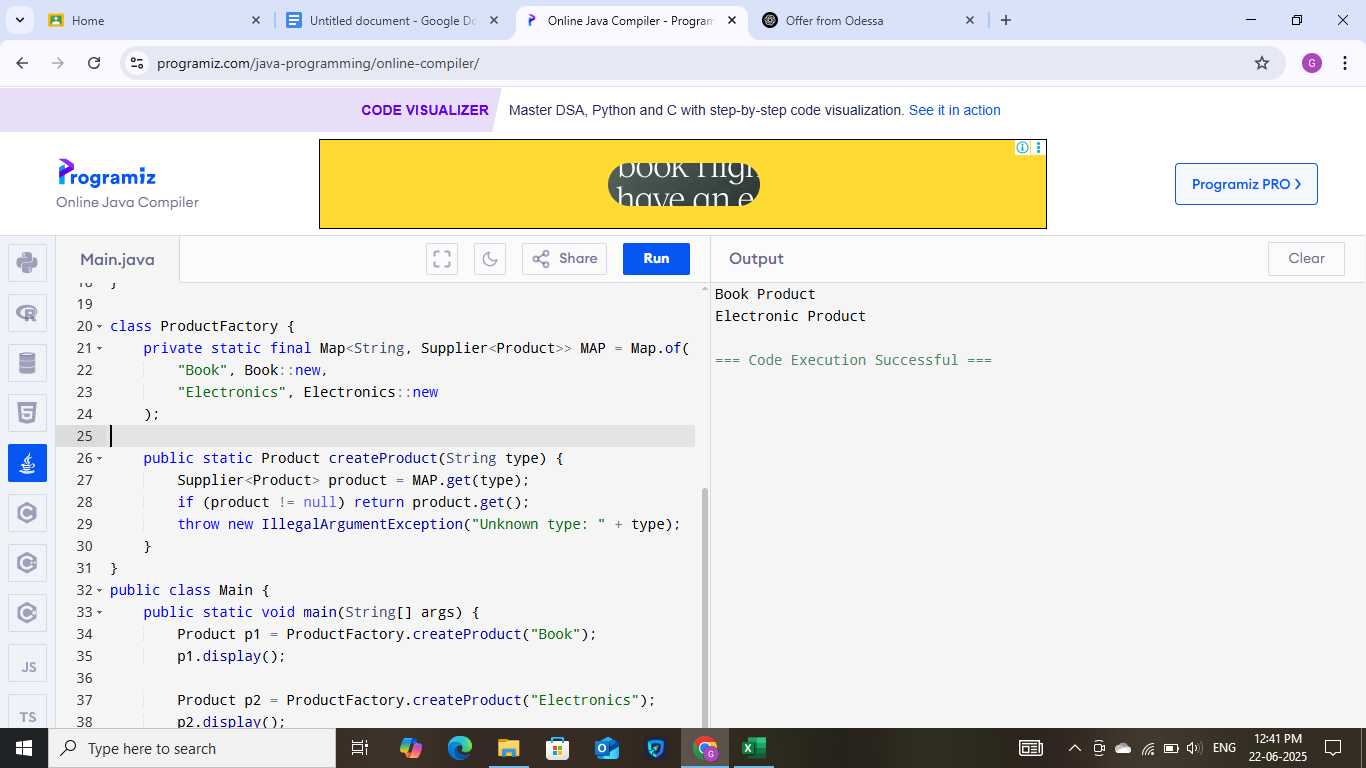
p1.display();

Product p2 = ProductFactory.createProduct("Electronics");

p2.display();

}

}



**Exercise 3: E-commerce Platform Search Function**

**Coding:**

import java.util.\*;

import java.util.stream.\*;

public class ProductSearch {

public static List<String> searchProducts(List<String> products, String keyword) {

String regex = ".\*" + keyword.toLowerCase() + ".\*";

return products.stream()

.filter(p -> p.toLowerCase().matches(regex))

.collect(Collectors.toList());

}

public static void main(String[] args) {

List<String> products = Arrays.asList("Laptop", "Smartphone", "Lamp", "Smartwatch", "Table");

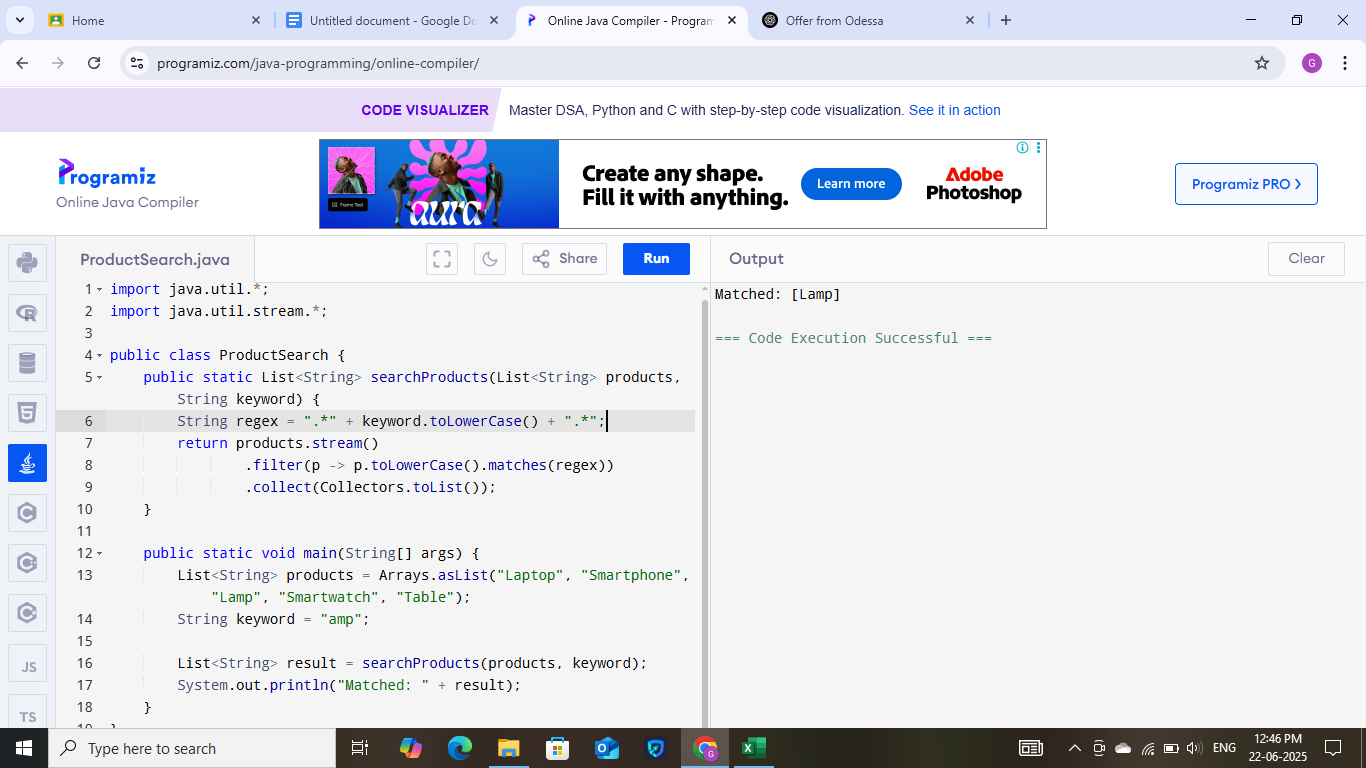
String keyword = "amp";

List<String> result = searchProducts(products, keyword);

System.out.println("Matched: " + result);

}

}



**Exercise 4: Financial Forecasting**

**Coding:**

import java.util.\*;

public class FinancialForecasting {

public static double forecastRevenue(int[] revenue, int windowSize) {

if (revenue.length < windowSize) return -1;

Deque<Integer> window = new LinkedList<>();

int sum = 0;

for (int i = 0; i < revenue.length; i++) {

window.addLast(revenue[i]);

sum += revenue[i];

if (window.size() > windowSize) {

sum -= window.removeFirst();

}

}

return sum / (double) windowSize;

}

public static void main(String[] args) {

int[] revenue = {1000, 1200, 1500, 1700, 2000};

int windowSize = 3;

double forecast = forecastRevenue(revenue, windowSize);

System.out.printf("Forecast: %.2f%n", forecast);

}

}

