**WEEK 01**

**Exercise 1: Implementing the Singleton Pattern**

public class ConfigManager {

private static ConfigManager instance;

private ConfigManager() {}

public static ConfigManager getInstance() {

if (instance == null) {

synchronized (ConfigManager.class) {

if (instance == null) {

instance = new ConfigManager();

}

}

}

return instance;

}

public void loadSettings() {

System.out.println("Settings loaded.");

}

public static void main(String[] args) {

ConfigManager config1 = ConfigManager.getInstance();

ConfigManager config2 = ConfigManager.getInstance();

System.out.println(config1 == config2);

config1.loadSettings();

}

}

**OUTPUT :**



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

interface Notification {

void notifyUser();

}

class SMSNotification implements Notification {

public void notifyUser() {

System.out.println("Sending an SMS Notification");

}

}

class EmailNotification implements Notification {

public void notifyUser() {

System.out.println("Sending an Email Notification");

}

}

class PushNotification implements Notification {

public void notifyUser() {

System.out.println("Sending a Push Notification");

}

}

abstract class NotificationFactory {

public abstract Notification createNotification();

}

class SMSNotificationFactory extends NotificationFactory {

public Notification createNotification() {

return new SMSNotification();

}

}

class EmailNotificationFactory extends NotificationFactory {

public Notification createNotification() {

return new EmailNotification();

}

}

class PushNotificationFactory extends NotificationFactory {

public Notification createNotification() {

return new PushNotification();

}

}

public class Main {

public static void main(String[] args) {

NotificationFactory factory = new SMSNotificationFactory();

Notification notification = factory.createNotification();

notification.notifyUser();

factory = new EmailNotificationFactory();

notification = factory.createNotification();

notification.notifyUser();

factory = new PushNotificationFactory();

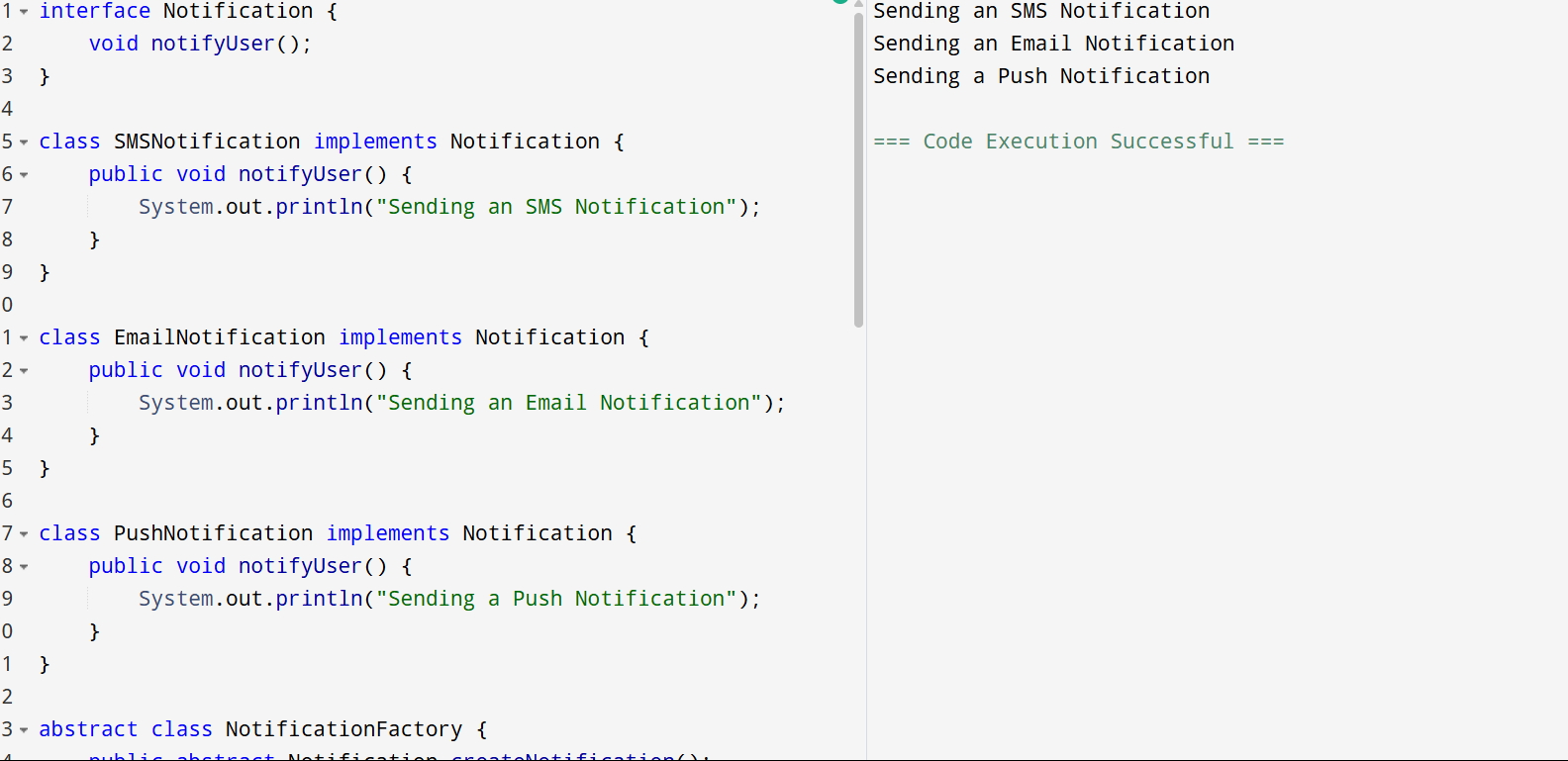
notification = factory.createNotification();

notification.notifyUser();

}

}

**OUTPUT :**



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

import java.util.\*;

public class ECommerceSearch {

public static void main(String[] args) {

SearchEngine engine = new SearchEngine();

engine.addProduct("laptop");

engine.addProduct("laptop case");

engine.addProduct("laptop charger");

engine.addProduct("phone");

engine.addProduct("phone case");

engine.addProduct("headphones");

List<String> results = engine.search("lap");

System.out.println("Search results for 'lap': " + results);

}

}

class TrieNode {

Map<Character, TrieNode> children = new HashMap<>();

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

boolean isEnd = false;

}

class SearchEngine {

TrieNode root = new TrieNode();

void addProduct(String product) {

TrieNode node = root;

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

node.words.add(product);

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

}

node.words.add(product);

node.isEnd = true;

}

List<String> search(String prefix) {

TrieNode node = root;

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

if (!node.children.containsKey(c)) return new ArrayList<>();

node = node.children.get(c);

}

return new ArrayList<>(node.words);

}

}

**OUTPUT :**



**Exercise 7: Financial Forecasting**

import java.util.\*;

public class FinancialForecasting {

public static void main(String[] args) {

int[] revenue = {1200, 1500, 1300, 1600, 1700, 1800, 1900, 2100, 2300, 2500, 2400, 2600};

int[] forecast = movingAverageForecast(revenue, 3);

System.out.println("Historical Revenue: " + Arrays.toString(revenue));

System.out.println("Forecasted Revenue: " + Arrays.toString(forecast));

int trend = detectTrend(revenue);

System.out.println("Revenue Trend: " + (trend > 0 ? "Increasing" : trend < 0 ? "Decreasing" : "Stable"));

int maxProfit = maxProfitInterval(revenue);

System.out.println("Maximum Profit Interval: " + maxProfit);

}

static int[] movingAverageForecast(int[] data, int window) {

int[] result = new int[data.length];

int sum = 0;

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

sum += data[i];

if (i >= window) sum -= data[i - window];

result[i] = i >= window - 1 ? sum / window : data[i];

}

return result;

}

static int detectTrend(int[] data) {

int n = data.length;

int up = 0, down = 0;

for (int i = 1; i < n; i++) {

if (data[i] > data[i - 1]) up++;

else if (data[i] < data[i - 1]) down++;

}

return Integer.compare(up, down);

}

static int maxProfitInterval(int[] revenue) {

int maxProfit = 0, minValue = revenue[0];

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

maxProfit = Math.max(maxProfit, revenue[i] - minValue);

minValue = Math.min(minValue, revenue[i]);

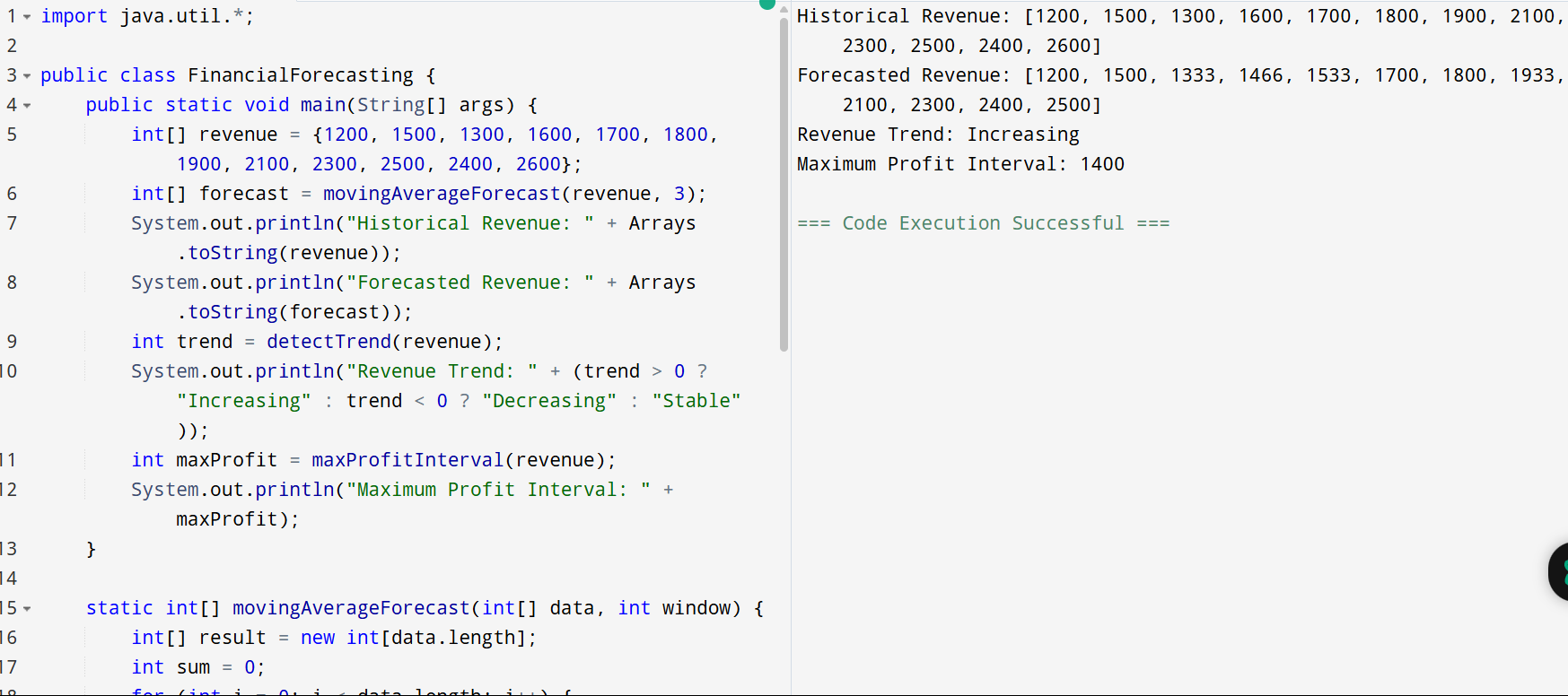
}

return maxProfit;

}

}

**OUTPUT :**

****