**COMPULSARY QNS FOR WEEK 1**

**DESIGN PATTERNS AND PRINCIPLES**

**1.SINGLETON METHOD**

**CODE:**

import java.util.Scanner;

class EventLogger {

private static EventLogger instance;

private EventLogger() {

}

public static EventLogger getInstance() {

if (instance == null) {

instance = new EventLogger();

}

return instance;

}

public void log(String message) {

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

}

}

public class Main {

public static void main(String[] args) {

EventLogger logger = EventLogger.getInstance();

Scanner scanner = new Scanner(System.in);

System.out.println("Enter number of log messages:");

int count = scanner.nextInt();

scanner.nextLine();

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

System.out.println("Enter log message " + i + ":");

String message = scanner.nextLine();

logger.log(message);

}

EventLogger anotherLogger = EventLogger.getInstance();

if (logger == anotherLogger) {

System.out.println("Only one EventLogger instance used.");

} else {

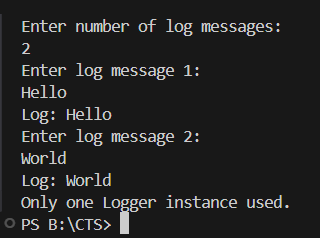
System.out.println("Multiple EventLogger instances detected.");

}

}

}

Output:



**2.FACTORY MRTHOD PATTERN**

**CODE:**

import java.util.Scanner;

interface Document {

void open();

}

class WordDocument implements Document {

public void open() {

System.out.println("Opening Word Document");

}

}

class PdfDocument implements Document {

public void open() {

System.out.println("Opening PDF Document");

}

}

class ExcelDocument implements Document {

public void open() {

System.out.println("Opening Excel Document");

}

}

abstract class DocumentFactory {

abstract Document createDocument();

}

class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter document type (word, pdf, excel):");

String input = scanner.nextLine().toLowerCase();

DocumentFactory factory;

switch (input) {

case "word":

factory = new WordDocumentFactory();

break;

case "pdf":

factory = new PdfDocumentFactory();

break;

case "excel":

factory = new ExcelDocumentFactory();

break;

default:

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

return;

}

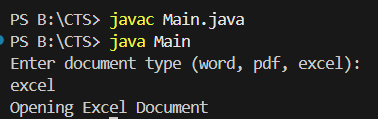
Document document = factory.createDocument();

document.open();

}

}

Output:



**PRACTICE QNS:**

**DESIGN PATTERNS AND PRINCIPLES**

**Proxypattern :**

**Code:**

import java.util.\*;

interface Image {

void display();

}

class RealImage implements Image {

private String filename;

public RealImage(String filename) {

this.filename = filename;

loadFromRemoteServer();

}

private void loadFromRemoteServer() {

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

}

@Override

public void display() {

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

}

}a

class ProxyImage implements Image {

private String filename;

private RealImage realImage;

public ProxyImage(String filename) {

this.filename = filename;

}

@Override

public void display() {

if (realImage == null) {

realImage = new RealImage(filename); // Lazy initialization

} else {

System.out.println("Using cached image for: " + filename);

}

realImage.display();

}

}

public class ProxyPatternExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Map<String, ProxyImage> imageCache = new HashMap<>();

System.out.println("Image Viewer Application (type 'exit' to quit)");

while (true) {

System.out.print("\nEnter image name to view: ");

String input = scanner.nextLine().trim();

if (input.equalsIgnoreCase("exit")) {

break;

}

ProxyImage image = imageCache.get(input);

if (image == null) {

image = new ProxyImage(input);

imageCache.put(input, image);

}

image.display();

}

scanner.close();

System.out.println("Exiting Image Viewer.");

}

}

Output:

