Assignment-1

Questions:

1. Create a database connection manager in Java using the Singleton design pattern. Show how this guarantees that the application uses a single instance.

Ans)

Code:

class DatabaseConnectionManager {

private static DatabaseConnectionManager instance;

private DatabaseConnectionManager() {

// Private constructor to prevent instantiation

}

public static DatabaseConnectionManager getInstance() {

if (instance == null) {

synchronized (DatabaseConnectionManager.class) {

if (instance == null) {

instance = new DatabaseConnectionManager();

}

}

}

return instance;

}

public void connect() {

System.out.println("Connected to the database.");

}

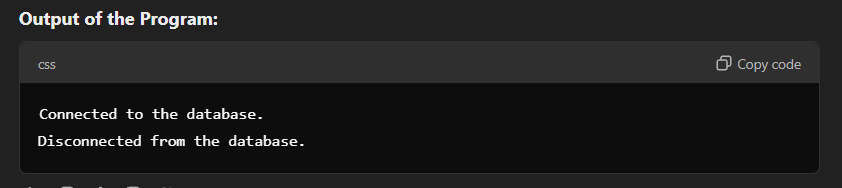
public void disconnect() {

System.out.println("Disconnected from the database.");

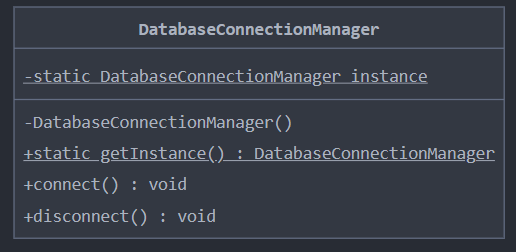
}

}

I/0:



Class Diagram:



1. Create e-commerce application examples using the Factory Method and Abstract Factory design patterns. Examine how they have implemented various payment methods.

Ans)

Code:

interface PaymentMethod {

void processPayment();

}

class CreditCardPayment implements PaymentMethod {

public void processPayment() {

System.out.println("Processing credit card payment.");

}

}

class DebitCardPayment implements PaymentMethod {

public void processPayment() {

System.out.println("Processing debit card payment.");

}

}

class PaymentFactory {

public static PaymentMethod getPaymentMethod(String type) {

switch (type.toLowerCase()) {

case "credit":

return new CreditCardPayment();

case "debit":

return new DebitCardPayment();

default:

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

}

}

}

interface PaymentGateway {

PaymentMethod createPaymentMethod(String type);

}

class StripePaymentGateway implements PaymentGateway {

public PaymentMethod createPaymentMethod(String type) {

return PaymentFactory.getPaymentMethod(type);

}

}

class PayPalPaymentGateway implements PaymentGateway {

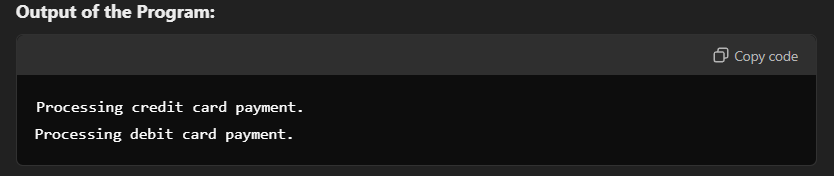
public PaymentMethod createPaymentMethod(String type) {

return PaymentFactory.getPaymentMethod(type);

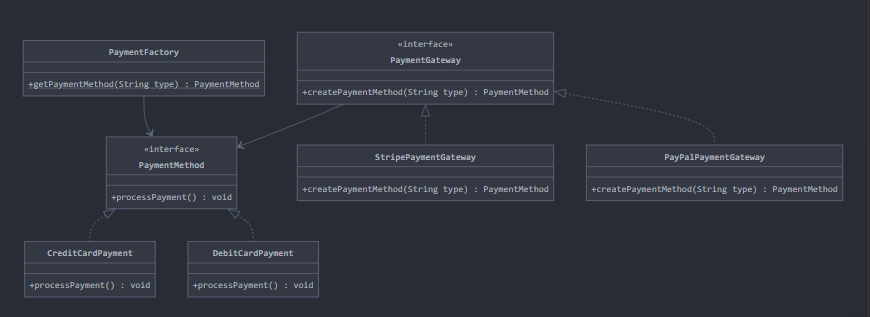
}

}

I/0:



Class diagram :



1. Use the Observer design pattern to model a stock market application in which several clients receive price alerts.

Ans)

Code:

import java.util.\*;

interface Observer {

void update(String stock, double price);

}

class StockMarket {

private Map<String, Double> stocks = new HashMap<>();

private List<Observer> observers = new ArrayList<>();

public void addObserver(Observer observer) {

observers.add(observer);

}

public void removeObserver(Observer observer) {

observers.remove(observer);

}

public void updateStock(String stock, double price) {

stocks.put(stock, price);

notifyObservers(stock, price);

}

private void notifyObservers(String stock, double price) {

for (Observer observer : observers) {

observer.update(stock, price);

}

}

}

class Client implements Observer {

private String name;

public Client(String name) {

this.name = name;

}

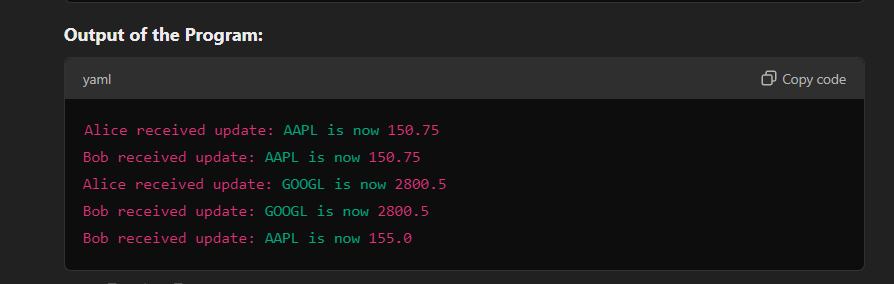
public void update(String stock, double price) {

System.out.println(name + " received update: " + stock + " is now " + price);

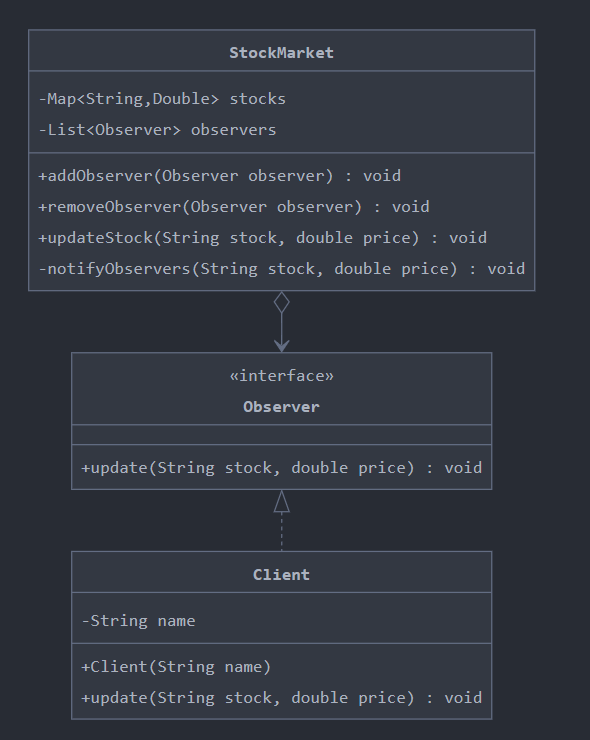
}

}

I/O:



Class Diagram:



1. Use the Abstract Factory pattern to create a payment processing system that supports several payment gateways, including Stripe and PayPal. Methods for initializing and validating transactions should be implemented by every gateway

Ans)

Code:

interface PaymentProcessorFactory {

PaymentProcessor createProcessor();

}

interface PaymentProcessor {

void initializeTransaction();

void validateTransaction();

}

class StripeProcessor implements PaymentProcessor {

public void initializeTransaction() {

System.out.println("Initializing transaction with Stripe.");

}

public void validateTransaction() {

System.out.println("Validating transaction with Stripe.");

}

}

class PayPalProcessor implements PaymentProcessor {

public void initializeTransaction() {

System.out.println("Initializing transaction with PayPal.");

}

public void validateTransaction() {

System.out.println("Validating transaction with PayPal.");

}

}

class StripeProcessorFactory implements PaymentProcessorFactory {

public PaymentProcessor createProcessor() {

return new StripeProcessor();

}

}

class PayPalProcessorFactory implements PaymentProcessorFactory {

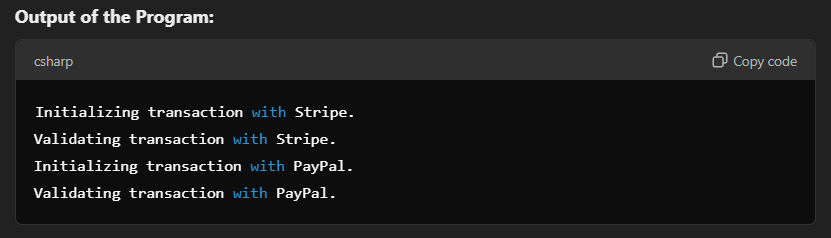
public PaymentProcessor createProcessor() {

return new PayPalProcessor();

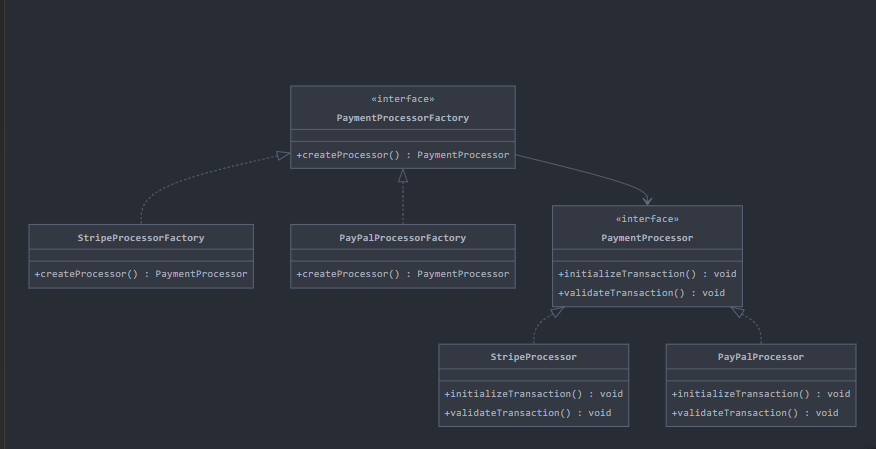
}

}

I/O:



Class Diagram:



1. Create a Java program that handles WordDocument, PDFDocument, and ExcelDocument documents by utilizing the Factory Pattern. To demonstrate how to use it in the client class, create a DocumentFactory that will instantiate the objects according to the document type.

Ans)

Code:

interface Document {

void open();

void save();

}

class WordDocument implements Document {

public void open() {

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

}

public void save() {

System.out.println("Saving Word document.");

}

}

class PDFDocument implements Document {

public void open() {

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

}

public void save() {

System.out.println("Saving PDF document.");

}

}

class ExcelDocument implements Document {

public void open() {

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

}

public void save() {

System.out.println("Saving Excel document.");

}

}

class DocumentFactory {

public static Document createDocument(String type) {

switch (type.toLowerCase()) {

case "word":

return new WordDocument();

case "pdf":

return new PDFDocument();

case "excel":

return new ExcelDocument();

default:

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

}

}

}

public class Main {

public static void main(String[] args) {

// Singleton Example

DatabaseConnectionManager manager = DatabaseConnectionManager.getInstance();

manager.connect();

manager.disconnect();

// Factory and Abstract Factory Examples

PaymentGateway stripeGateway = new StripePaymentGateway();

PaymentMethod stripePayment = stripeGateway.createPaymentMethod("credit");

stripePayment.processPayment();

PaymentGateway payPalGateway = new PayPalPaymentGateway();

PaymentMethod payPalPayment = payPalGateway.createPaymentMethod("debit");

payPalPayment.processPayment();

// Observer Example

StockMarket market = new StockMarket();

Client clientA = new Client("Client A");

Client clientB = new Client("Client B");

market.addObserver(clientA);

market.addObserver(clientB);

market.updateStock("AAPL", 150.00);

// Abstract Factory Example

PaymentProcessorFactory stripeFactory = new StripeProcessorFactory();

PaymentProcessor stripeProcessor = stripeFactory.createProcessor();

stripeProcessor.initializeTransaction();

stripeProcessor.validateTransaction();

// Document Factory Example

Document wordDoc = DocumentFactory.createDocument("word");

wordDoc.open();

wordDoc.save();

Document pdfDoc = DocumentFactory.createDocument("pdf");

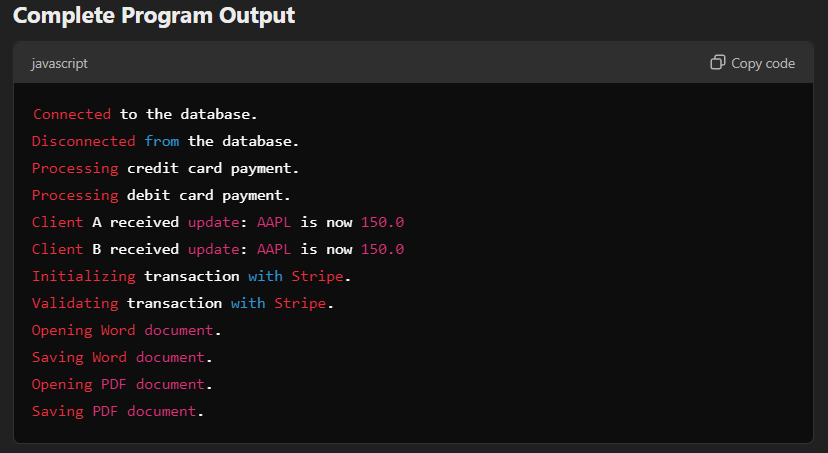
pdfDoc.open();

pdfDoc.save();

}

}

I/O:



Class Diagram:

