**Design Pattern and Principles (6427548 Hinayath Sulthana A)**

**1.Singleton Pattern**

package example;

public class LoggerExample {

interface Logger {

void log(String message);

}

static class SystemLogger implements Logger {

private static SystemLogger *instance*;

private SystemLogger() {}

public static SystemLogger getInstance() {

if (*instance* == null) {

*instance* = new SystemLogger();

}

return *instance*;

}

*@Override*

public void log(String message) {

System.*out*.println("[System] " + message);

}

*@Override*

public String toString() {

return "SystemLogger@" + Integer.*toHexString*(System.*identityHashCode*(this));

}

}

public static void main(String[] args) {

Logger systemLogger = SystemLogger.*getInstance*();

Logger anotherSystemLogger = SystemLogger.*getInstance*();

systemLogger.log("System initialized.");

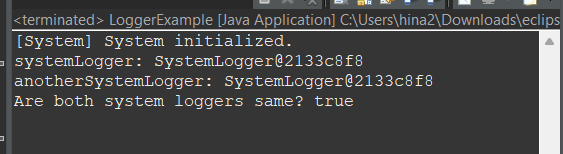
System.*out*.println("systemLogger: " + systemLogger);

System.*out*.println("anotherSystemLogger: " + anotherSystemLogger);

System.*out*.println("Are both system loggers same? " + (systemLogger == anotherSystemLogger));

}

}



1.O/P

**2. Factory Pattern**

**Manager.java**

package factor;

interface Document {

void view();

}

class Pdf implements Document {

public void view() {

System.*out*.println("Pdf Document is opened");

}

}

class Word implements Document {

public void view() {

System.*out*.println("Word Document is opened");

}

}

class Excel implements Document {

public void view() {

System.*out*.println("Excel Document is opened");

}

}

abstract class DocumentFactory {

public abstract Document createDocument();

}

class WordDocument extends DocumentFactory {

public Document createDocument() {

return new Word();

}

}

class PdfDocument extends DocumentFactory {

public Document createDocument() {

return new Pdf();

}

}

class ExcelDocument extends DocumentFactory {

public Document createDocument() {

return new Excel();

}

}

public class Manager {

public static void main(String[] args) {

DocumentFactory factory;

factory = new PdfDocument();

Document pdf = factory.createDocument();

pdf.view();

factory = new WordDocument();

Document word = factory.createDocument();

word.view();

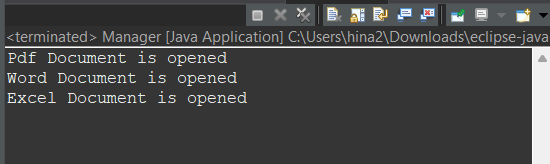
factory = new ExcelDocument();

Document excel = factory.createDocument();

excel.view();

}

}



2.O/P

**3.Builder Pattern**

**BuilderPatternTest.java:**

package builder;

public class BuilderPatternTest {

public static void main(String[] args) {

Computer gamingPC = new Computer.Builder()

.setCPU("Intel i9")

.setRAM("32GB")

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 4080")

.enableWiFi(true)

.enableBluetooth(true)

.build();

Computer officePC = new Computer.Builder()

.setCPU("Intel i5")

.setRAM("16GB")

.setStorage("512GB SSD")

.enableWiFi(false)

.enableBluetooth(false)

.build();

System.***out***.println("Gaming PC: " + gamingPC);

System.***out***.println("Office PC: " + officePC);

} }

**Computer.java:**

package builder;

public class Computer {

private String CPU;

private String RAM;

private String storage;

private String graphicsCard;

private boolean isWiFiEnabled;

private boolean isBluetoothEnabled;

private Computer(Builder builder) {

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.storage = builder.storage;

this.graphicsCard = builder.graphicsCard;

this.isWiFiEnabled = builder.isWiFiEnabled;

this.isBluetoothEnabled = builder.isBluetoothEnabled;

}

public String toString() {

return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", Storage=" + storage +

", GraphicsCard=" + graphicsCard + ", WiFi=" + isWiFiEnabled +

", Bluetooth=" + isBluetoothEnabled + "]";

}

public static class Builder {

private String CPU;

private String RAM;

private String storage;

private String graphicsCard;

private boolean isWiFiEnabled;

private boolean isBluetoothEnabled;

public Builder setCPU(String CPU) {

this.CPU = CPU;

return this;

}

public Builder setRAM(String RAM) {

this.RAM = RAM;

return this;

}

public Builder setStorage(String storage) {

this.storage = storage;

return this;

}

public Builder setGraphicsCard(String graphicsCard) {

this.graphicsCard = graphicsCard;

return this;

}

public Builder enableWiFi(boolean value) {

this.isWiFiEnabled = value;

return this;

}

public Builder enableBluetooth(boolean value) {

this.isBluetoothEnabled = value;

return this;

}

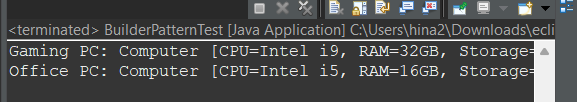
public Computer build() {

return new Computer(this);

}

}

}



3.O/P

**4.Adapter Pattern**

**AdapterPatternExample.java:**

package examplee;

interface PaymentProcessor {

void processPayment(double amount);

}

class PayPal {

public void makePayPalPayment(double amount) {

System.***out***.println("Paid " + amount + " using PayPal.");

}

}

class Stripe {

public void stripePay(double amount, String currency) {

System.***out***.println("Paid " + amount + " " + currency + " using Stripe.");

}

}

class PayPalAdapter implements PaymentProcessor {

private PayPal payPal;

public PayPalAdapter(PayPal payPal) {

this.payPal = payPal;

}

*@Override*

public void processPayment(double amount) {

payPal.makePayPalPayment(amount);

}

}

class StripeAdapter implements PaymentProcessor {

private Stripe stripe;

public StripeAdapter(Stripe stripe) {

this.stripe = stripe;

}

*@Override*

public void processPayment(double amount) {

stripe.stripePay(amount, "INR");

}

}

public class AdapterPatternExample {

public static void main(String[] args) {

PaymentProcessor payPalPayment = new PayPalAdapter(new PayPal());

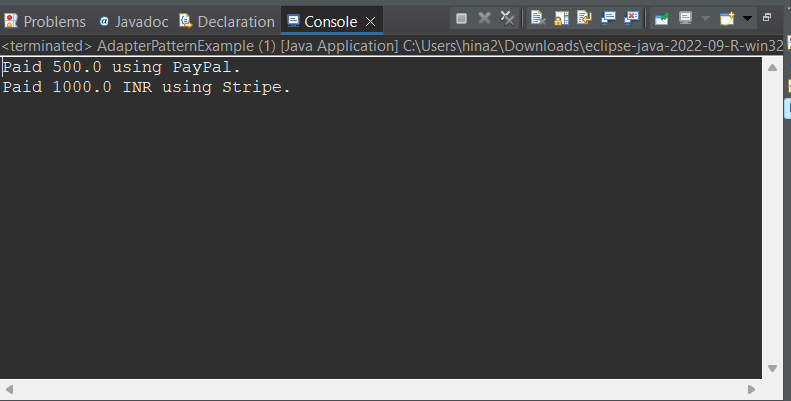
payPalPayment.processPayment(500.0);

PaymentProcessor stripePayment = new StripeAdapter(new Stripe());

stripePayment.processPayment(1000.0);

}

}



4.O/P

**5. Decorator Pattern**

**Main.java:**

package decoratorexample;

public class Main {

public static void main(String args[]) {

Notifier notifier = new EmailNotifier();

notifier = new SMSNotifierDecorator(notifier);

notifier.send("System update at 6 PM");

}

}

**EmailNotifier.java:**

package decoratorexample;

public class EmailNotifier implements Notifier {

*@Override*

public void send(String message) {

System.*out*.println("Sending Email: "+message);

}

}

**Notifier.java:**

package decoratorexample;

public interface Notifier {

void send(String message);

}

**NotifierDecorator.java:**

package decoratorexample;

public class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier=notifier;

}

public void send(String message) {

wrappedNotifier.send(message);

}

}

**SMSNotifierDecorator.java:**

package decoratorexample;

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

*@Override*

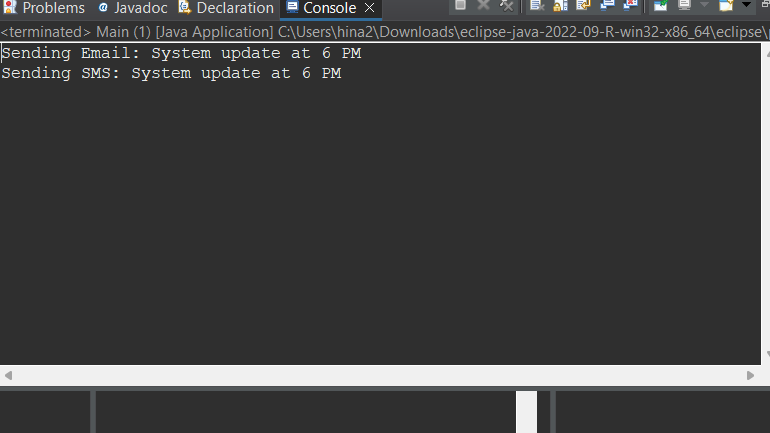
public void send(String message) {

super.send(message);

System.***out***.println("Sending SMS: "+message);

}

}



5.O/P

**6. Proxy Pattern**

**Image.java**

package proxy;

public interface Image {

void display();

}

**ProxyImage.java**

package proxy;

public 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);

}

realImage.display();

}

}

**ProxyPatternExample.java:**

package proxy;

public class ProxyPatternExample {

public static void main(String args[]) {

Image image1=new ProxyImage("cat\_photo.jpg");

System.***out***.println("First call:");

image1.display();

System.***out***.println("\nSecond call:");

image1.display();

}

}

**Realimage.java:**

package proxy;

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);

}

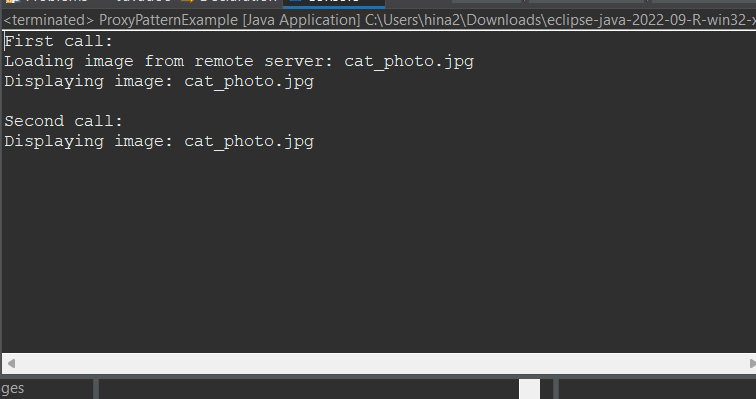
*@Override*

public void display() {

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

}

}



6.O/P

**7.Observer Pattern**

**MobileApp.java:**

package observerexp;

public class MobileApp implements Observer{

private String name;

public MobileApp(String name) {

this.name = name;

}

*@Override*

public void update(String stockName, double price) {

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

}

}

**Observer.java:**

package observerexp;

interface Observer {

void update(String stockName, double price);

}

**ObserverPatternExample.java:**

package observerexp;

public class ObserverPatternExample {

public static void main(String[] args) {

StockMarket stockMarket = new StockMarket();

Observer mobile1 = new MobileApp("Alice");

Observer mobile2 = new MobileApp("Bob");

Observer web = new WebApp("InvestPro"); // This must be valid

stockMarket.registerObserver(mobile1);

stockMarket.registerObserver(mobile2);

stockMarket.registerObserver(web);

stockMarket.setStock("AAPL", 210.5);

}

}

**StockMarket.java:**

package observerexp;

import java.util.List;

import java.util.ArrayList;

public class StockMarket implements Stock {

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

private String stockName;

private double stockPrice;

public void setStock(String name, double price) {

this.stockName = name;

this.stockPrice = price;

notifyObservers();

}

*@Override*

public void registerObserver(Observer o) {

observers.add(o);

}

*@Override*

public void removeObserver(Observer o) {

observers.remove(o);

}

*@Override*

public void notifyObservers() {

for (Observer o : observers) {

o.update(stockName, stockPrice);

}

}

}

**Stock.java:**

package observerexp;

public interface Stock {

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObservers();

}

**WebApp.java:**

package observerexp;

public class WebApp implements Observer {

private String name;

public WebApp(String name) {

this.name = name;

}

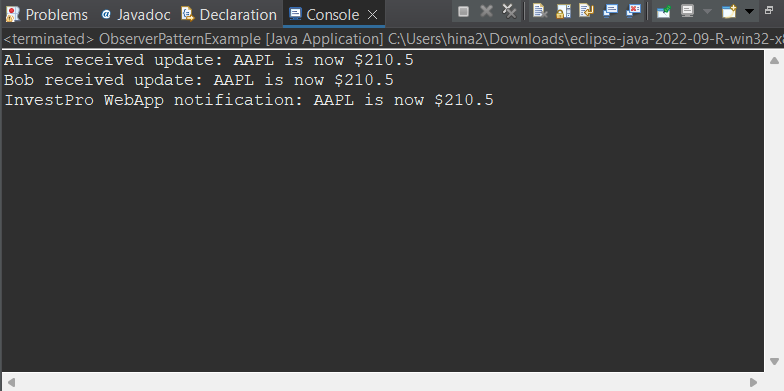
*@Override*

public void update(String stockName, double price) {

System.***out***.println(name + " WebApp notification: " + stockName + " is now $" + price);

}

}



7.O/P

**8 . Strategy Pattern**

**CreditCardPayment.java:**

package strategyexp;

class CreditCardPayment implements PaymentStrategy {

private String cardNumber;

private String cardHolder;

public CreditCardPayment(String cardNumber, String cardHolder) {

this.cardNumber = cardNumber;

this.cardHolder = cardHolder;

}

*@Override*

public void pay(double amount) {

System.***out***.println("Paid $" + amount + " using Credit Card (" + cardHolder + ")");

}

}

**PaymentContext.java:**

package strategyexp;

class PaymentContext{

private PaymentStrategy strategy;

public void setPaymentStrategy(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void payAmount(double amount) {

if (strategy == null) {

System.***out***.println("Payment strategy not set.");

} else {

strategy.pay(amount);

}

}

}

**PaymentStrategy.java:**

package strategyexp;

public interface PaymentStrategy {

void pay(double amount);

}

**PayPalPayment.java:**

package strategyexp;

class PayPalPayment implements PaymentStrategy {

private String email;

public PayPalPayment(String email) {

this.email = email;

}

*@Override*

public void pay(double amount) {

System.***out***.println("Paid $" + amount + " using PayPal (" + email + ")");

}

}

**StrategyPatternExample:**

package strategyexp;

public class StrategyPatternExample {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

PaymentStrategy creditCard = new CreditCardPayment("1234-5678-9012", "Alice");

context.setPaymentStrategy(creditCard);

context.payAmount(150.00);

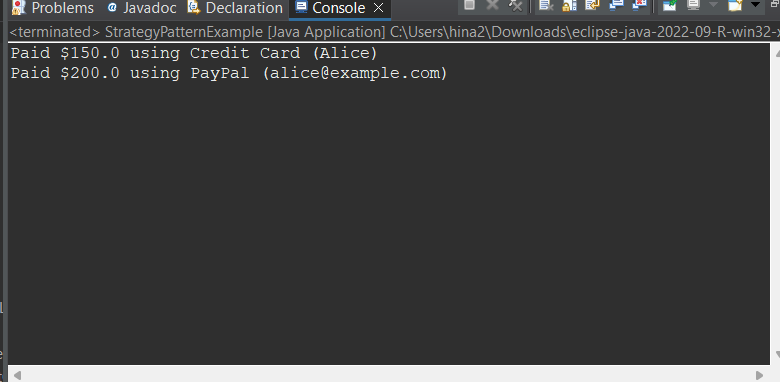
PaymentStrategy paypal = new PayPalPayment("alice@example.com");

context.setPaymentStrategy(paypal);

context.payAmount(200.00);

}

}

****

8. O/P

**9. Command Pattern**

**Command.java:**

package commandexp;

public interface Command {

void execute();

}

**CommandPatternExample.java:**

package commandexp;

public class CommandPatternExample {

public static void main(String[] args) {

Light livingRoomLight = new Light("Living Room");

Command lightsOn = new LightOnCommand(livingRoomLight);

Command lightsOff = new LightOffCommand(livingRoomLight);

RemoteControl remote = new RemoteControl();

remote.setCommand(lightsOn);

remote.pressButton();

remote.setCommand(lightsOff);

remote.pressButton();

}

}

**Light.java:**

package commandexp;

public class Light {

private String location;

public Light(String location) {

this.location = location;

}

public void turnOn() {

System.***out***.println(location + " light is ON");

}

public void turnOff() {

System.***out***.println(location + " light is OFF");

}

}

**LightOffCommand.java:**

package commandexp;

public class Light {

private String location;

public Light(String location) {

this.location = location;

}

public void turnOn() {

System.***out***.println(location + " light is ON");

}

public void turnOff() {

System.***out***.println(location + " light is OFF");

} }

**LightOnCommand.java:**

package commandexp;

public class Light {

private String location;

public Light(String location) {

this.location = location;

}

public void turnOn() {

System.***out***.println(location + " light is ON");

}

public void turnOff() {

System.***out***.println(location + " light is OFF");

}

}

**RemoteControl.java:**

package commandexp;

public class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

if (command != null) {

command.execute();

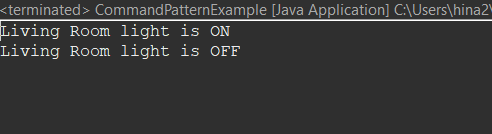
} else {

System.***out***.println("No command set.");

}

}

}



**10. MVC Pattern**

**MVCDemo.java:**

package mvcexp;

public class MVCDemo {

public static void main(String[] args) {

Student student = new Student();

student.setName("Hina");

student.setRollNo("22CS003");

StudentView view = new StudentView();

StudentController controller = new StudentController(student, view);

controller.updateView();

controller.setStudentName("Hinayath Sulthana");

controller.updateView();

}

}

**Student.java:**

package mvcexp;

public class Student {

private String name;

private String rollNo;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getRollNo() {

return rollNo;

}

public void setRollNo(String rollNo) {

this.rollNo = rollNo;

}

}

**StudentController.java:**

package mvcexp;

public class StudentController {

private Student model;

private StudentView view;

public StudentController(Student model, StudentView view) {

this.model = model;

this.view = view;

}

public void setStudentName(String name) {

model.setName(name);

}

public void setStudentRollNo(String rollNo) {

model.setRollNo(rollNo);

}

public String getStudentName() {

return model.getName();

}

public String getStudentRollNo() {

return model.getRollNo();

}

public void updateView() {

view.printStudentDetails(model.getName(), model.getRollNo());

}

}

**StudentView.java:**

package mvcexp;

public class StudentView {

public void printStudentDetails(String name, String rollNo) {

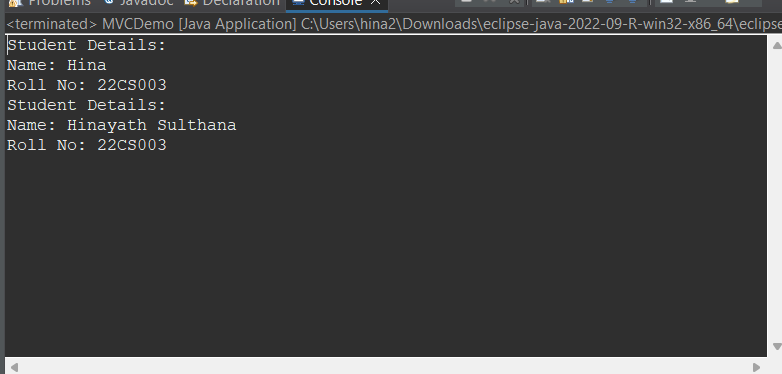
System.***out***.println("Student Details:");

System.***out***.println("Name: " + name);

System.***out***.println("Roll No: " + rollNo);

}

}



10.O/P

**11. Dependency Injection**

**Customer.java:**

package dependentexp;

public class Customer {

private int id;

private String name;

public Customer(int id, String name) {

this.id = id;

this.name = name;

}

public String getName() {

return name;

}

public int getId() {

return id;

}

}

**CustomerService.java:**

package dependentexp;

public class CustomerService {

private CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public void getCustomerInfo(int id) {

Customer customer = customerRepository.findCustomerById(id);

System.***out***.println("Customer found: ID = " + customer.getId() + ", Name = " + customer.getName());

}

}

**CustomerRepisotiry.java:**

package dependentexp;

public interface CustomerRepository {

Customer findCustomerById(int id);

}

**DIDemo.java:**

package dependentexp;

public class DIDemo {

public static void main(String[] args) {

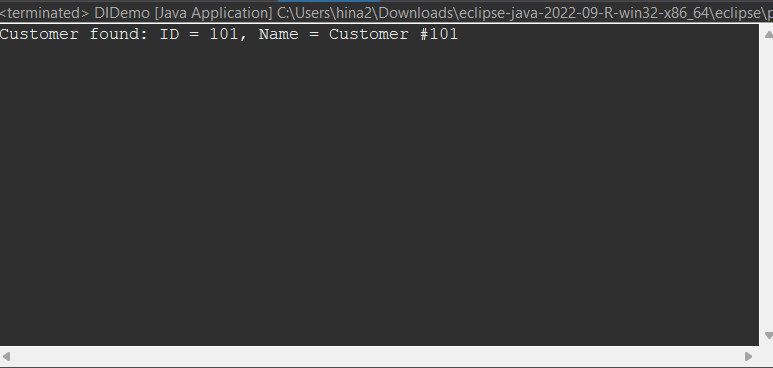
CustomerRepository repository = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repository);

service.getCustomerInfo(101);

}

}



11.O/P