**Exercise 1: Singleton Pattern**

**Logger.java**

public class Logger {

private static Logger instance;

private Logger() {

System.out.println("Logger Initialized");

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

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

}

}

**TestLogger.java**

public class TestLogger {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("First Message");

System.out.println(logger1 == logger2); // true

}

}

**Exercise 2: Factory Method Pattern**

**Document.java**

public interface Document {

void open();

}

**WordDocument.java / PdfDocument.java**

public class WordDocument implements Document {

public void open() { System.out.println("Opening Word Document"); }

}

public class PdfDocument implements Document {

public void open() { System.out.println("Opening PDF Document"); }

}

**DocumentFactory.java**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordFactory.java / PdfFactory.java**

public class WordFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

public class PdfFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**TestFactory.java**

public class TestFactory {

public static void main(String[] args) {

DocumentFactory factory = new WordFactory();

Document doc = factory.createDocument();

doc.open();

}

}

**Exercise 3: Builder Pattern**

**Computer.java**

public class Computer {

private String cpu, ram, storage;

private Computer(Builder builder) {

this.cpu = builder.cpu;

this.ram = builder.ram;

this.storage = builder.storage;

}

public static class Builder {

private String cpu, ram, storage;

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 Computer build() {

return new Computer(this);

}

}

public void showSpecs() {

System.out.println("CPU: " + cpu + ", RAM: " + ram + ", Storage: " + storage);

}

}

**TestBuilder.java**

public class TestBuilder {

public static void main(String[] args) {

Computer pc = new Computer.Builder()

.setCPU("i7")

.setRAM("16GB")

.setStorage("1TB SSD")

.build();

pc.showSpecs();

}

}

**Exercise 4: Adapter Pattern**

**PaymentProcessor.java**

public interface PaymentProcessor {

void processPayment(double amount);

}

**PayPalGateway.java**

java

CopyEdit

public class PayPalGateway {

public void makePayment(double amount) {

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

}

}

**PayPalAdapter.java**

public class PayPalAdapter implements PaymentProcessor {

private PayPalGateway gateway = new PayPalGateway();

public void processPayment(double amount) {

gateway.makePayment(amount);

}

}

**TestAdapter.java**

public class TestAdapter {

public static void main(String[] args) {

PaymentProcessor processor = new PayPalAdapter();

processor.processPayment(500);

}

}

**Exercise 5: Decorator Pattern**

**Notifier.java**

public interface Notifier {

void send(String message);

}

**EmailNotifier.java**

java

CopyEdit

public class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

**NotifierDecorator.java / SMSNotifierDecorator.java**

public abstract class NotifierDecorator implements Notifier {

protected Notifier notifier;

public NotifierDecorator(Notifier notifier) {

this.notifier = notifier;

}

public void send(String message) {

notifier.send(message);

}

}

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

**TestDecorator.java**

public class TestDecorator {

public static void main(String[] args) {

Notifier notifier = new SMSNotifierDecorator(new EmailNotifier());

notifier.send("Hello!");

}

}

**Exercise 6: Proxy Pattern**

**Image.java**

public interface Image {

void display();

}

**RealImage.java / ProxyImage.java**

java

CopyEdit

public class RealImage implements Image {

private String filename;

public RealImage(String filename) {

this.filename = filename;

loadImage();

}

private void loadImage() {

System.out.println("Loading " + filename);

}

public void display() {

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

}

}

public class ProxyImage implements Image {

private RealImage realImage;

private String filename;

public ProxyImage(String filename) {

this.filename = filename;

}

public void display() {

if (realImage == null)

realImage = new RealImage(filename);

realImage.display();

}

}

**TestProxy.java**

public class TestProxy {

public static void main(String[] args) {

Image img = new ProxyImage("photo.jpg");

img.display();

img.display();

}

}

**Exercise 7: Observer Pattern**

**Observer.java / Stock.java**

java

CopyEdit

public interface Observer {

void update(float price);

}

public interface Stock {

void register(Observer o);

void deregister(Observer o);

void notifyObservers();

}

**StockMarket.java / MobileApp.java**

import java.util.\*;

public class StockMarket implements Stock {

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

private float price;

public void register(Observer o) { observers.add(o); }

public void deregister(Observer o) { observers.remove(o); }

public void setPrice(float price) {

this.price = price;

notifyObservers();

}

public void notifyObservers() {

for (Observer o : observers) {

o.update(price);

}

}

}

public class MobileApp implements Observer {

public void update(float price) {

System.out.println("Updated price: " + price);

}

}

**TestObserver.java**

public class TestObserver {

public static void main(String[] args) {

StockMarket market = new StockMarket();

Observer app = new MobileApp();

market.register(app);

market.setPrice(150.0f);

}

}

**Exercise 8: Strategy Pattern**

**PaymentStrategy.java**

public interface PaymentStrategy {

void pay(double amount);

}

**CreditCardPayment.java**

public class CreditCardPayment implements PaymentStrategy {

public void pay(double amount) {

System.out.println("Paid " + amount + " via Credit Card");

}

}

**PaymentContext.java**

public class PaymentContext {

private PaymentStrategy strategy;

public void setStrategy(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void pay(double amount) {

strategy.pay(amount);

}

}

**TestStrategy.java**

public class TestStrategy {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

context.setStrategy(new CreditCardPayment());

context.pay(999);

}

}

**Exercise 9: Command Pattern**

**Command.java / Light.java**

public interface Command {

void execute();

}

public class Light {

public void on() { System.out.println("Light ON"); }

public void off() { System.out.println("Light OFF"); }

}

**LightOnCommand.java / RemoteControl.java**

public class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) { this.light = light; }

public void execute() {

light.on();

}

}

public class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

**TestCommand.java**

public class TestCommand {

public static void main(String[] args) {

Light light = new Light();

Command cmd = new LightOnCommand(light);

RemoteControl remote = new RemoteControl();

remote.setCommand(cmd);

remote.pressButton();

}

}

**Exercise 10: MVC Pattern**

**Student.java**

public class Student {

private String name, id, grade;

public Student(String name, String id, String grade) {

this.name = name; this.id = id; this.grade = grade;

}

public String getName() { return name; }

public String getGrade() { return grade; }

public void setName(String name) { this.name = name; }

}

**StudentView.java**

public class StudentView {

public void displayStudentDetails(Student s) {

System.out.println("Student: " + s.getName() + ", Grade: " + s.getGrade());

}

}

**StudentController.java**

public class StudentController {

private Student model;

private StudentView view;

public StudentController(Student model, StudentView view) {

this.model = model;

this.view = view;

}

public void updateView() {

view.displayStudentDetails(model);

}

public void setName(String name) {

model.setName(name);

}

}

**MVCTest.java**

public class MVCTest {

public static void main(String[] args) {

Student s = new Student("Alice", "001", "A+");

StudentView v = new StudentView();

StudentController c = new StudentController(s, v);

c.updateView();

c.setName("Alicia");

c.updateView();

}

}

**Exercise 11: Dependency Injection**

**CustomerRepository.java / CustomerRepositoryImpl.java**

public interface CustomerRepository {

String findCustomerById(String id);

}

public class CustomerRepositoryImpl implements CustomerRepository {

public String findCustomerById(String id) {

return "Customer ID: " + id;

}

}

**CustomerService.java**

public class CustomerService {

private CustomerRepository repo;

public CustomerService(CustomerRepository repo) {

this.repo = repo;

}

public void getCustomer(String id) {

System.out.println(repo.findCustomerById(id));

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

CustomerRepository repo = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repo);

service.getCustomer("123");

}

}