Java Design Pattern Implementations

# Singleton Pattern

## Logger.java

package singleton;  
  
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);  
 }  
}

## Main.java

package singleton;  
  
public class Main {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.getInstance();  
 logger1.log("First log message");  
 Logger logger2 = Logger.getInstance();  
 logger2.log("Second log message");  
 System.out.println("Same instance? " + (logger1 == logger2));  
 }  
}

# Factory Pattern

## Main.java

package factory;  
  
public class Main {  
 public static void main(String[] args) {  
 DocumentFactory factory = new PdfFactory();  
 Document doc = factory.createDocument();  
 doc.open();  
 }  
}

## PdfFactory.java

package factory;  
  
public class PdfFactory extends DocumentFactory {  
 public Document createDocument() {  
 return new PdfDocument();  
 }  
}

## PdfDocument.java

package factory;  
  
public class PdfDocument implements Document {  
 public void open() {  
 System.out.println("Opening PDF document...");  
 }  
}

## DocumentFactory.java

package factory;  
  
public abstract class DocumentFactory {  
 public abstract Document createDocument();  
}

## Document.java

package factory;  
  
public interface Document {  
 void open();  
}

# Builder Pattern

## Computer.java

package builder;  
  
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 display() {  
 System.out.println("Computer [CPU=" + CPU + ", RAM=" + RAM + ", Storage=" + storage + "]");  
 }  
}

## Main.java

package builder;  
  
public class Main {  
 public static void main(String[] args) {  
 Computer comp = new Computer.Builder()  
 .setCPU("Intel i7")  
 .setRAM("16GB")  
 .setStorage("1TB SSD")  
 .build();  
 comp.display();  
 }  
}

# Adapter Pattern

## PaymentProcessor.java

package adapter;  
  
public interface PaymentProcessor {  
 void processPayment(double amount);  
}

## PayPalGateway.java

package adapter;  
  
public class PayPalGateway {  
 public void sendPayment(double amount) {  
 System.out.println("PayPal paid: $" + amount);  
 }  
}

## PayPalAdapter.java

package adapter;  
  
public class PayPalAdapter implements PaymentProcessor {  
 private PayPalGateway gateway;  
 public PayPalAdapter(PayPalGateway gateway) {  
 this.gateway = gateway;  
 }  
 public void processPayment(double amount) {  
 gateway.sendPayment(amount);  
 }  
}

## Main.java

package adapter;  
  
public class Main {  
 public static void main(String[] args) {  
 PaymentProcessor processor = new PayPalAdapter(new PayPalGateway());  
 processor.processPayment(500);  
 }  
}

# Decorator Pattern

## Notifier.java

package decorator;  
  
public interface Notifier {  
 void send(String message);  
}

## EmailNotifier.java

package decorator;  
  
public class EmailNotifier implements Notifier {  
 public void send(String message) {  
 System.out.println("Email: " + message);  
 }  
}

## SMSNotifierDecorator.java

package decorator;  
  
public class SMSNotifierDecorator implements Notifier {  
 private Notifier notifier;  
 public SMSNotifierDecorator(Notifier notifier) { this.notifier = notifier; }  
 public void send(String message) {  
 notifier.send(message);  
 System.out.println("SMS: " + message);  
 }  
}

## Main.java

package decorator;  
  
public class Main {  
 public static void main(String[] args) {  
 Notifier notifier = new SMSNotifierDecorator(new EmailNotifier());  
 notifier.send("Alert!");  
 }  
}

# Proxy Pattern

package proxy;

public interface Image {

void display();

}

package proxy;

public class RealImage implements Image {

private String filename;

public RealImage(String filename) {

this.filename = filename;

loadImage();

}

private void loadImage() {

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

}

public void display() {

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

}

}

package proxy;

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();

}

}

package proxy;

public class Main {

public static void main(String[] args) {

Image image = new ProxyImage("test.jpg");

image.display();

System.out.println("Displaying again...");

image.display();

}

}

# Observer Pattern

package observer;

public interface Observer {

void update(String stock, double price);

}

package observer;

import java.util.\*;

public class StockMarket {

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

private String stock;

private double price;

public void register(Observer o) {

observers.add(o);

}

public void setStock(String stock, double price) {

this.stock = stock;

this.price = price;

notifyObservers();

}

public void notifyObservers() {

for (Observer o : observers) {

o.update(stock, price);

}

}

}

package observer;

public class MobileApp implements Observer {

public void update(String stock, double price) {

System.out.println("Mobile App - " + stock + ": " + price);

}

}

package observer;

public class Main {

public static void main(String[] args) {

StockMarket market = new StockMarket();

Observer mobile = new MobileApp();

market.register(mobile);

market.setStock("AAPL", 150.0);

}

}

# Strategy Pattern

package strategy;

public interface PaymentStrategy {

void pay(double amount);

}

package strategy;

public class CreditCardPayment implements PaymentStrategy {

public void pay(double amount) {

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

}

}

package strategy;

public class PaymentContext {

private PaymentStrategy strategy;

public void setStrategy(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void pay(double amount) {

strategy.pay(amount);

}

}

package strategy;

public class Main {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

context.setStrategy(new CreditCardPayment());

context.pay(100);

context.setStrategy(new PayPalPayment());

context.pay(200);

}

}

# Command Pattern

package command;

public interface Command {

void execute();

}

package command;

public class Light {

public void on() {

System.out.println("Light ON");

}

public void off() {

System.out.println("Light OFF");

}

}

package command;

public class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.on();

}

}

package command;

public class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

package command;

public class Main {

public static void main(String[] args) {

Light light = new Light();

RemoteControl remote = new RemoteControl();

remote.setCommand(new LightOnCommand(light));

remote.pressButton();

remote.setCommand(new LightOffCommand(light));

remote.pressButton();

}

}

# MVC Pattern

package mvc;

public class StudentView {

public void displayStudentDetails(Student student) {

System.out.println("Student: " + student.getName() + ", ID: " + student.getId() + ", Grade: " + student.getGrade());

}

}

package mvc;

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 setStudentGrade(String grade) {

model.setGrade(grade);

}

public void updateView() {

view.displayStudentDetails(model);

}

}

package mvc;

public class Main {

public static void main(String[] args) {

Student student = new Student("Alice", "101", "A");

StudentView view = new StudentView();

StudentController controller = new StudentController(student, view);

controller.updateView();

controller.setStudentGrade("A+");

controller.updateView();

}

}

# Dependency Injection Pattern

package di;

public interface CustomerRepository {

String findCustomerById(String id);

}

package di;

public class CustomerRepositoryImpl implements CustomerRepository {

public String findCustomerById(String id) {

return "Customer: " + id;

}

}

package di;

public class CustomerService {

private CustomerRepository repo;

public CustomerService(CustomerRepository repo) {

this.repo = repo;

}

public void findCustomer(String id) {

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

}

}

package di;

public class Main {

public static void main(String[] args) {

CustomerRepository repo = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repo);

service.findCustomer("C123");

}

}