Design Patterns and Principles - Java Implementations

# 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);  
 }  
}

// Main.java

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

# Exercise 2: Factory Method Pattern

// Document.java

interface Document {  
 void open();  
}

// WordDocument.java

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

// PdfDocument.java

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

// DocumentFactory.java

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

// PdfFactory.java

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

// Main.java

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

# Exercise 3: Builder Pattern

// Computer.java

public class Computer {  
 private String cpu;  
 private String ram;  
 private String storage;  
  
 private Computer(Builder builder) {  
 this.cpu = builder.cpu;  
 this.ram = builder.ram;  
 this.storage = builder.storage;  
 }  
  
 public static class Builder {  
 private String cpu;  
 private String ram;  
 private String 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 displaySpecs() {  
 System.out.println("CPU: " + cpu + ", RAM: " + ram + ", Storage: " + storage);  
 }  
}

// Main.java

public class Main {  
 public static void main(String[] args) {  
 Computer pc = new Computer.Builder()  
 .setCpu("Intel i7")  
 .setRam("16GB")  
 .setStorage("512GB SSD")  
 .build();  
  
 pc.displaySpecs();  
 }  
}

# Exercise 4: Adapter Pattern

// PaymentProcessor.java

interface PaymentProcessor {  
 void processPayment(double amount);  
}

// PayPalService.java

class PayPalService {  
 void sendMoney(double amount) {  
 System.out.println("Sending $" + amount + " via PayPal");  
 }  
}

// PayPalAdapter.java

class PayPalAdapter implements PaymentProcessor {  
 private PayPalService payPal;  
  
 public PayPalAdapter() {  
 this.payPal = new PayPalService();  
 }  
  
 public void processPayment(double amount) {  
 payPal.sendMoney(amount);  
 }  
}

// Main.java

public class Main {  
 public static void main(String[] args) {  
 PaymentProcessor processor = new PayPalAdapter();  
 processor.processPayment(250.0);  
 }  
}

# Exercise 5: Decorator Pattern

// Notifier.java

interface Notifier {  
 void send(String message);  
}

// EmailNotifier.java

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

// NotifierDecorator.java

abstract class NotifierDecorator implements Notifier {  
 protected Notifier notifier;  
  
 public NotifierDecorator(Notifier notifier) {  
 this.notifier = notifier;  
 }  
  
 public void send(String message) {  
 notifier.send(message);  
 }  
}

// SMSNotifier.java

class SMSNotifier extends NotifierDecorator {  
 public SMSNotifier(Notifier notifier) {  
 super(notifier);  
 }  
  
 public void send(String message) {  
 super.send(message);  
 System.out.println("SMS: " + message);  
 }  
}

// Main.java

public class Main {  
 public static void main(String[] args) {  
 Notifier notifier = new SMSNotifier(new EmailNotifier());  
 notifier.send("Server is down!");  
 }  
}

# Exercise 6: Proxy Pattern

// Image.java

interface Image {  
 void display();  
}

// RealImage.java

class RealImage implements Image {  
 private String filename;  
 public RealImage(String filename) {  
 this.filename = filename;  
 loadImageFromDisk();  
 }  
  
 private void loadImageFromDisk() {  
 System.out.println("Loading " + filename);  
 }  
  
 public void display() {  
 System.out.println("Displaying " + filename);  
 }  
}

// ProxyImage.java

class ProxyImage implements Image {  
 private String filename;  
 private RealImage realImage;  
  
 public ProxyImage(String filename) {  
 this.filename = filename;  
 }  
  
 public void display() {  
 if (realImage == null) {  
 realImage = new RealImage(filename);  
 }  
 realImage.display();  
 }  
}

// Main.java

public class Main {  
 public static void main(String[] args) {  
 Image img = new ProxyImage("photo.jpg");  
 img.display();  
 img.display();  
 }  
}

# Exercise 7: Observer Pattern

// Observer.java

interface Observer {  
 void update(float price);  
}

// Stock.java

interface Stock {  
 void register(Observer obs);  
 void unregister(Observer obs);  
 void notifyObservers();  
}

// StockMarket.java

import java.util.\*;  
  
class StockMarket implements Stock {  
 private List<Observer> observers = new ArrayList<>();  
 private float price;

public void setPrice(float price) {  
 this.price = price;  
 notifyObservers();  
 }  
  
 public void register(Observer obs) {  
 observers.add(obs);  
 }  
  
 public void unregister(Observer obs) {  
 observers.remove(obs);  
 }  
  
 public void notifyObservers() {  
 for (Observer obs : observers) {  
 obs.update(price);  
 }  
 }  
}

// MobileApp.java

class MobileApp implements Observer {  
 public void update(float price) {  
 System.out.println("Mobile App: Stock price changed to " + price);  
 }  
}

// Main.java

public class Main {  
 public static void main(String[] args) {  
 StockMarket market = new StockMarket();  
 Observer mobile = new MobileApp();  
  
 market.register(mobile);  
 market.setPrice(101.5f);  
 }  
}

**Exercise 8: Strategy Pattern**

// PaymentStrategy.java

interface PaymentStrategy {

void pay(double amount);

}

// CreditCardPayment.java

class CreditCardPayment implements PaymentStrategy {

public void pay(double amount) {

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

}

}

// PaymentContext.java

class PaymentContext {

private PaymentStrategy strategy;

public PaymentContext(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void makePayment(double amount) {

strategy.pay(amount);

}

}

// Main.java

public class Main {

public static void main(String[] args) {

PaymentStrategy strategy = new CreditCardPayment();

PaymentContext context = new PaymentContext(strategy);

context.makePayment(500.0);

}

}

**Exercise 9: Command Pattern**

// Command.java

interface Command {

void execute();

}

// Light.java

class Light {

public void turnOn() {

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

}

public void turnOff() {

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

}

}

// LightOnCommand.java

class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOn();

}

}

// RemoteControl.java

class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

// Main.java

public class Main {

public static void main(String[] args) {

Light light = new Light();

Command lightOn = new LightOnCommand(light);

RemoteControl remote = new RemoteControl();

remote.setCommand(lightOn);

remote.pressButton();

}

}

**Exercise 10: MVC Pattern**

// Student.java

class Student {

private String name;

private String id;

private String 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 getId() { return id; }

public String getGrade() { return grade; }

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

public void setGrade(String grade) { this.grade = grade; }

}

// StudentView.java

class StudentView {

public void displayDetails(Student student) {

System.out.println("Name: " + student.getName());

System.out.println("ID: " + student.getId());

System.out.println("Grade: " + student.getGrade());

}

}

// StudentController.java

class StudentController {

private Student student;

private StudentView view;

public StudentController(Student student, StudentView view) {

this.student = student;

this.view = view;

}

public void updateView() {

view.displayDetails(student);

}

public void changeName(String name) {

student.setName(name);

}

public void changeGrade(String grade) {

student.setGrade(grade);

}

}

// Main.java

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.changeGrade("A+");

controller.updateView();

}

}

**Exercise 11: Dependency Injection**

// CustomerRepository.java

interface CustomerRepository {

String findCustomerById(String id);

}

// CustomerRepositoryImpl.java

class CustomerRepositoryImpl implements CustomerRepository {

public String findCustomerById(String id) {

return "Customer ID: " + id + ", Name: John Doe";

}

}

// CustomerService.java

class CustomerService {

private CustomerRepository repository;

public CustomerService(CustomerRepository repository) {

this.repository = repository;

}

public void getCustomer(String id) {

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

}

}

// Main.java

public class Main {

public static void main(String[] args) {

CustomerRepository repo = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repo);

service.getCustomer("C101");

}

}