Design Patterns and Principles - Solutions

## Exercise 1: Singleton Pattern

public class Logger {  
 private static Logger instance;  
  
 private Logger() {  
 // private constructor  
 }  
  
 public static Logger getInstance() {  
 if (instance == null) {  
 instance = new Logger();  
 }  
 return instance;  
 }  
  
 public void log(String message) {  
 System.out.println(message);  
 }  
}  
  
public class SingletonTest {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.getInstance();  
 Logger logger2 = Logger.getInstance();  
 logger1.log("Logger test");  
  
 System.out.println(logger1 == logger2); // true  
 }  
}

## Exercise 2: Factory Method Pattern

interface Document {  
 void open();  
}  
  
class WordDocument implements Document {  
 public void open() { System.out.println("Opening Word Document"); }  
}  
  
class PdfDocument implements Document {  
 public void open() { System.out.println("Opening PDF Document"); }  
}  
  
class ExcelDocument implements Document {  
 public void open() { System.out.println("Opening Excel Document"); }  
}  
  
abstract class DocumentFactory {  
 public abstract Document createDocument();  
}  
  
class WordFactory extends DocumentFactory {  
 public Document createDocument() { return new WordDocument(); }  
}  
  
class PdfFactory extends DocumentFactory {  
 public Document createDocument() { return new PdfDocument(); }  
}  
  
class ExcelFactory extends DocumentFactory {  
 public Document createDocument() { return new ExcelDocument(); }  
}  
  
public class FactoryTest {  
 public static void main(String[] args) {  
 DocumentFactory factory = new PdfFactory();  
 Document doc = factory.createDocument();  
 doc.open();  
 }  
}

## Exercise 3: Builder Pattern

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 String toString() {  
 return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", Storage=" + storage + "]";  
 }  
}  
  
public class BuilderTest {  
 public static void main(String[] args) {  
 Computer comp = new Computer.Builder().setCPU("i7").setRAM("16GB").setStorage("1TB").build();  
 System.out.println(comp);  
 }  
}

## Exercise 4: Adapter Pattern

interface PaymentProcessor {  
 void processPayment(double amount);  
}  
  
class PayPalGateway {  
 public void sendPayment(double amount) {  
 System.out.println("Processing PayPal payment of $" + amount);  
 }  
}  
  
class PayPalAdapter implements PaymentProcessor {  
 private PayPalGateway gateway = new PayPalGateway();  
  
 public void processPayment(double amount) {  
 gateway.sendPayment(amount);  
 }  
}  
  
public class AdapterTest {  
 public static void main(String[] args) {  
 PaymentProcessor processor = new PayPalAdapter();  
 processor.processPayment(100.0);  
 }  
}

## Exercise 5: Decorator Pattern

interface Notifier {  
 void send(String message);  
}  
  
class EmailNotifier implements Notifier {  
 public void send(String message) {  
 System.out.println("Email: " + message);  
 }  
}  
  
abstract class NotifierDecorator implements Notifier {  
 protected Notifier notifier;  
  
 public NotifierDecorator(Notifier notifier) {  
 this.notifier = notifier;  
 }  
}  
  
class SMSNotifier extends NotifierDecorator {  
 public SMSNotifier(Notifier notifier) {  
 super(notifier);  
 }  
  
 public void send(String message) {  
 notifier.send(message);  
 System.out.println("SMS: " + message);  
 }  
}  
  
public class DecoratorTest {  
 public static void main(String[] args) {  
 Notifier notifier = new SMSNotifier(new EmailNotifier());  
 notifier.send("System Alert!");  
 }  
}

## Exercise 6: Proxy Pattern

interface Image {  
 void display();  
}  
  
class RealImage implements Image {  
 private String filename;  
  
 public RealImage(String filename) {  
 this.filename = filename;  
 loadFromDisk();  
 }  
  
 private void loadFromDisk() {  
 System.out.println("Loading " + filename);  
 }  
  
 public void display() {  
 System.out.println("Displaying " + filename);  
 }  
}  
  
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();  
 }  
}  
  
public class ProxyTest {  
 public static void main(String[] args) {  
 Image image = new ProxyImage("photo.jpg");  
 image.display(); // load + display  
 image.display(); // only display  
 }  
}

## Exercise 7: Observer Pattern

import java.util.\*;  
  
interface Observer {  
 void update(String stock, double price);  
}  
  
interface Stock {  
 void register(Observer o);  
 void deregister(Observer o);  
 void notifyObservers();  
}  
  
class StockMarket implements Stock {  
 private List<Observer> observers = new ArrayList<>();  
 private String stock;  
 private double price;  
  
 public void setStockData(String stock, double price) {  
 this.stock = stock;  
 this.price = price;  
 notifyObservers();  
 }  
  
 public void register(Observer o) { observers.add(o); }  
 public void deregister(Observer o) { observers.remove(o); }  
 public void notifyObservers() {  
 for (Observer o : observers) o.update(stock, price);  
 }  
}  
  
class MobileApp implements Observer {  
 public void update(String stock, double price) {  
 System.out.println("Mobile App: " + stock + " updated to " + price);  
 }  
}  
  
public class ObserverTest {  
 public static void main(String[] args) {  
 StockMarket market = new StockMarket();  
 Observer mobile = new MobileApp();  
 market.register(mobile);  
 market.setStockData("AAPL", 189.50);  
 }  
}

## Exercise 8: Strategy Pattern

interface PaymentStrategy {  
 void pay(double amount);  
}  
  
class CreditCardPayment implements PaymentStrategy {  
 public void pay(double amount) {  
 System.out.println("Paid $" + amount + " using Credit Card.");  
 }  
}  
  
class PayPalPayment implements PaymentStrategy {  
 public void pay(double amount) {  
 System.out.println("Paid $" + amount + " using PayPal.");  
 }  
}  
  
class PaymentContext {  
 private PaymentStrategy strategy;  
  
 public PaymentContext(PaymentStrategy strategy) {  
 this.strategy = strategy;  
 }  
  
 public void executePayment(double amount) {  
 strategy.pay(amount);  
 }  
}  
  
public class StrategyTest {  
 public static void main(String[] args) {  
 PaymentContext context = new PaymentContext(new CreditCardPayment());  
 context.executePayment(150.0);  
  
 context = new PaymentContext(new PayPalPayment());  
 context.executePayment(75.0);  
 }  
}

## Exercise 9: Command Pattern

interface Command {  
 void execute();  
}  
  
class Light {  
 public void on() {  
 System.out.println("Light is ON");  
 }  
 public void off() {  
 System.out.println("Light is OFF");  
 }  
}  
  
class LightOnCommand implements Command {  
 private Light light;  
  
 public LightOnCommand(Light light) {  
 this.light = light;  
 }  
  
 public void execute() {  
 light.on();  
 }  
}  
  
class LightOffCommand implements Command {  
 private Light light;  
  
 public LightOffCommand(Light light) {  
 this.light = light;  
 }  
  
 public void execute() {  
 light.off();  
 }  
}  
  
class RemoteControl {  
 private Command command;  
  
 public void setCommand(Command command) {  
 this.command = command;  
 }  
  
 public void pressButton() {  
 command.execute();  
 }  
}  
  
public class CommandTest {  
 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();  
 }  
}

## Exercise 10: MVC Pattern

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 void setName(String name) { this.name = name; }  
  
 public String getId() { return id; }  
 public String getGrade() { return grade; }  
}  
  
class StudentView {  
 public void displayStudentDetails(Student student) {  
 System.out.println("Student: " + student.getName() + ", ID: " + student.getId() + ", Grade: " + student.getGrade());  
 }  
}  
  
class StudentController {  
 private Student model;  
 private StudentView view;  
  
 public StudentController(Student model, StudentView view) {  
 this.model = model;  
 this.view = view;  
 }  
  
 public void updateName(String name) {  
 model.setName(name);  
 }  
  
 public void updateView() {  
 view.displayStudentDetails(model);  
 }  
}  
  
public class MVCTest {  
 public static void main(String[] args) {  
 Student model = new Student("Alice", "S123", "A");  
 StudentView view = new StudentView();  
 StudentController controller = new StudentController(model, view);  
  
 controller.updateView();  
 controller.updateName("Bob");  
 controller.updateView();  
 }  
}

## Exercise 11: Dependency Injection

interface CustomerRepository {  
 String findCustomerById(String id);  
}  
  
class CustomerRepositoryImpl implements CustomerRepository {  
 public String findCustomerById(String id) {  
 return "Customer: " + id;  
 }  
}  
  
class CustomerService {  
 private CustomerRepository repository;  
  
 public CustomerService(CustomerRepository repository) {  
 this.repository = repository;  
 }  
  
 public void displayCustomer(String id) {  
 System.out.println(repository.findCustomerById(id));  
 }  
}  
  
public class DIExample {  
 public static void main(String[] args) {  
 CustomerRepository repo = new CustomerRepositoryImpl();  
 CustomerService service = new CustomerService(repo);  
 service.displayCustomer("C001");  
 }  
}