**Design Patterns and Principles – Handson Exercises**

**Exercise 3: Implementing the Builder Pattern**

**Code:**

// Computer.java

**package** BuilderPattern;

**public** **class** Computer{

**private** String CPU;

**private** **int** RAM;

**private** **int** 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** **int** RAM;

**private** **int** storage;

**public** Builder setCPU(String CPU){

**this**.CPU = CPU;

**return** **this**;

}

**public** Builder setRAM(**int** RAM){

**this**.RAM = RAM;

**return** **this**;

}

**public** Builder setStorage(**int** storage){

**this**.storage = storage;

**return** **this**;

}

**public** Computer build(){

**return** **new** Computer(**this**);

}

}

**public** **void** specs(){

System.***out***.printf("CPU:%s,RAM:%dGB,Storage:%dGB\n", CPU, RAM, storage);

}

}

// ComputerTest.java

**package** BuilderPattern;

**public** **class** ComputerTest{

**public** **static** **void** main(String[] args){

Computer gamingPC = **new** Computer.Builder()

.setCPU("Intel i3")

.setRAM(32)

.setStorage(1024)

.build();

gamingPC.specs();

Computer officePC = **new** Computer.Builder()

.setCPU("Intel i5")

.setRAM(16)

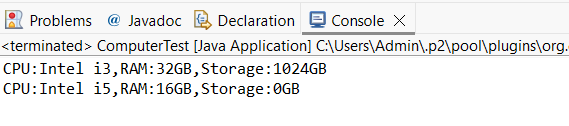
.build();

officePC.specs();

}

}

**Output:**



**Exercise 4: Implementing the Adapter Pattern**

**Code:**

// PaymentProcessor.java

**package** AdapterPattern;

**public** **interface** PaymentProcessor{

**void** processPayment(**double** amount);

}

**class** StripeGateway{

**public** **void** makePayment(**double** amt){

System.***out***.println("Stripe(processing payment) Rs."+amt);

}

}

**class** PayPalGateway{

**public** **void** sendPayment(**double** amt){

System.***out***.println("PayPal(sending payment) Rs."+amt);

}

}

// StripeAdapter.java

**package** AdapterPattern;

**public** **class** StripeAdapter **implements** PaymentProcessor{

**private** StripeGateway stripe = **new** StripeGateway();

**public** **void** processPayment(**double** amount){

stripe.makePayment(amount);

}

}

// PayPalAdapter.java

**package** AdapterPattern;

**public** **class** PayPalAdapter **implements** PaymentProcessor{

**private** PayPalGateway paypal = **new** PayPalGateway();

**public** **void** processPayment(**double** amount){

paypal.sendPayment(amount);

}

}

// PaymentTest.java

**package** AdapterPattern;

**public** **class** PaymentTest{

**public** **static** **void** main(String[] args){

PaymentProcessor stripe = **new** StripeAdapter();

PaymentProcessor paypal = **new** PayPalAdapter();

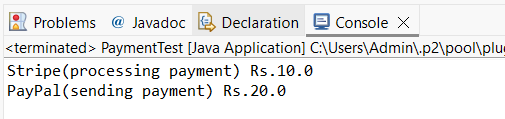
stripe.processPayment(10);

paypal.processPayment(20);

}

}

**Output:**



**Exercise 5: Implementing the Decorator Pattern**

**Code:**

// EmailNotifier.java

**package** DecoratorPattern;

**public** **class** EmailNotifier **implements** Notifier{

**public** **void** send(String message){

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

}

}

// Notifier.java

**package** DecoratorPattern;

**public** **interface** Notifier{

**void** send(String message);

}

//NotifierDecorator.java

**package** DecoratorPattern;

**public** **abstract** **class** NotifierDecorator **implements** Notifier{

**protected** Notifier wrappee;

**public** NotifierDecorator(Notifier notifier){

**this**.wrappee = notifier;

}

**public** **void** send(String message){

wrappee.send(message);

}

}

// SlackNotifierDecorator.java

**package** DecoratorPattern;

**public** **class** SlackNotifierDecorator **extends** NotifierDecorator{

**public** SlackNotifierDecorator(Notifier notifier){

**super**(notifier);

}

**public** **void** send(String message){

**super**.send(message);

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

}

}

// SMSNotifierDecorator.java

**package** DecoratorPattern;

**public** **class** SMSNotifierDecorator **extends** NotifierDecorator{

**public** SMSNotifierDecorator(Notifier notifier){

**super**(notifier);

}

**public** **void** send(String message){

**super**.send(message);

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

}

}

// NotificationTest.java

**package** DecoratorPattern;

**public** **class** NotificationTest{

**public** **static** **void** main(String[] args){

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

notifier = **new** SMSNotifierDecorator(notifier);

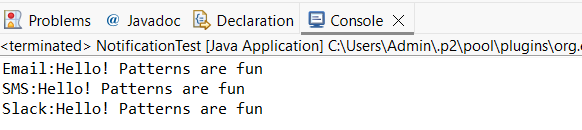
notifier = **new** SlackNotifierDecorator(notifier);

notifier.send("Hello! Patterns are fun");

}

}

**Output:**



**Exercise 6: Implementing the Proxy Pattern**

**Code:**

// Image.java

**package** ProxyPattern;

**public** **interface** Image{

**void** display();

}

// RealImage.java

**package** ProxyPattern;

**public** **class** RealImage **implements** Image{

**private** String filename;

**public** RealImage(String filename){

**this**.filename = filename;

loadFromServer();

}

**private** **void** loadFromServer(){

System.***out***.println("Loading " + filename + " from server..");

}

**public** **void** display(){

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

}

}

// ProxyImage.java

**package** ProxyPattern;

**public** **class** ProxyImage **implements** Image{

**private** String filename;

**private** RealImage real;

**public** ProxyImage(String filename){

**this**.filename = filename;

}

**public** **void** display(){

**if** (real == **null**){

real = **new** RealImage(filename);

}

real.display();

}

}

// ImageViewerTest.java

**package** ProxyPattern;

**public** **class** ImageViewerTest{

**public** **static** **void** main(String[] args){

Image img = **new** ProxyImage("hello.jpg");

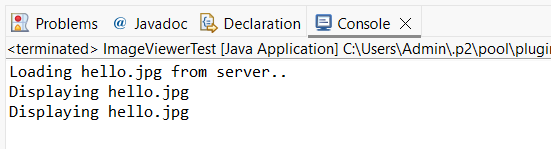
img.display();

img.display();

}

}

**Output:**



**Exercise 7: Implementing the Observer Pattern**

**Code:**

// Subject.java

**package** ObserverPattern;

**public** **interface** Subject{

**void** register(Observer o);

**void** deregister(Observer o);

**void** notifyObservers();

}

// StockMarket.java

**package** ObserverPattern;

**import** java.util.\*;

**public** **class** StockMarket **implements** Subject{

**private** Map<String, Double> stockPrices = **new** HashMap<>();

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

**public** **void** setPrice(String symbol, **double** price){

stockPrices.put(symbol, 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){

stockPrices.forEach((sym, pr) -> o.update(sym, pr));

}

}

}

// MobileApp.java

**package** ObserverPattern;

**public** **class** MobileApp **implements** Observer{

**public** **void** update(String stockSymbol, **double** price){

System.***out***.println("MobileApp:" + stockSymbol + " = Rs." + price);

}

}

// WebApp.java

**package** ObserverPattern;

**public** **class** WebApp **implements** Observer{

**public** **void** update(String stockSymbol, **double** price){

System.***out***.println("WebApp:" + stockSymbol + " = Rs." + price);

}

}

// StockTest.java

**package** ObserverPattern;

**public** **class** StockTest{

**public** **static** **void** main(String[] args){

StockMarket market = **new** StockMarket();

Observer mobile = **new** MobileApp();

Observer web = **new** WebApp();

market.register(mobile);

market.register(web);

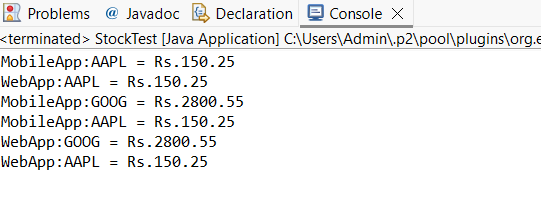
market.setPrice("AAPL", 150.25);

market.setPrice("GOOG", 2800.55);

}

}

**Output:**



**Exercise 8: Implementing the Strategy Pattern**

**Code:**

// PaymentStrategy.java

**package** StrategyPattern;

**public** **interface** PaymentStrategy{

**void** pay(**double** amount);

}

// CreditCardPayment.java

**package** StrategyPattern;

**public** **class** CreditCardPayment **implements** PaymentStrategy{

**public** **void** pay(**double** amount){

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

}

}

//PayPalPayment.java

**package** StrategyPattern;

**public** **class** PayPalPayment **implements** PaymentStrategy{

**public** **void** pay(**double** amount){

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

}

}

// PaymentContext.java

**package** StrategyPattern;

**public** **class** PaymentContext{

**private** PaymentStrategy strategy;

**public** PaymentContext(PaymentStrategy strategy){

**this**.strategy = strategy;

}

**public** **void** executePayment(**double** amount){

strategy.pay(amount);

}

}

// PaymentDemo.java

**package** StrategyPattern;

**public** **class** PaymentDemo{

**public** **static** **void** main(String[] args){

PaymentContext ctx = **new** PaymentContext(**new** CreditCardPayment());

ctx.executePayment(120.0);

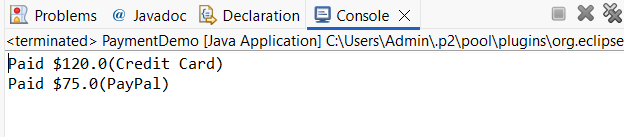
ctx = **new** PaymentContext(**new** PayPalPayment());

ctx.executePayment(75.0);

}

}

**Output:**



**Exercise 9: Implementing the Command Pattern**

**Code:**

// Command.java

**package** CommandPattern;

**public** **interface** Command {

**void** execute();

}

// Light.java

**package** CommandPattern;

**public** **class** Light{

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

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

}

// LightOnCommand.java

**package** CommandPattern;

**public** **class** LightOnCommand **implements** Command{

**private** Light light;

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

**public** **void** execute(){ light.on(); }

}

// LightOffCommand.java

**package** CommandPattern;

**public** **class** LightOffCommand **implements** Command{

**private** Light light;

**public** LightOffCommand(Light light){ **this**.light = light; }

**public** **void** execute(){ light.off(); }

}

// RemoteControl.java

**package** CommandPattern;

**public** **class** RemoteControl{

**private** Command command;

**public** **void** setCommand(Command command){ **this**.command = command; }

**public** **void** pressButton(){ command.execute(); }

}

// CommandTest.java

**package** CommandPattern;

**public** **class** CommandTest{

**public** **static** **void** main(String[] args){

Light livingRoomLight = **new** Light();

RemoteControl remote = **new** RemoteControl();

Command onCmd = **new** LightOnCommand(livingRoomLight);

Command offCmd = **new** LightOffCommand(livingRoomLight);

remote.setCommand(onCmd);

remote.pressButton();

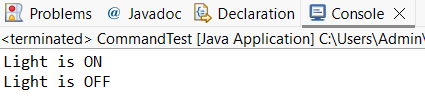
remote.setCommand(offCmd);

remote.pressButton();

}

}

**Output:**



**Exercise 10: Implementing the MVC Pattern**

**Code:**

//Student.java

**package** MVCPattern;

**public** **class** Student{

**private** String name;

**private** **int** id;

**private** **double** grade;

**public** Student(**int** id, String name, **double** grade){

**this**.id = id;

**this**.name = name;

**this**.grade = grade;

}

**public** String getName(){ **return** name; }

**public** **int** getId(){ **return** id; }

**public** **double** getGrade(){ **return** grade; }

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

}

// StudentView.java

**package** MVCPattern;

**public** **class** StudentView {

**public** **void** displayStudentDetails(**int** id, String name, **double** grade) {

System.***out***.println("Student ID: " + id);

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

System.***out***.println("Grade: " + grade);

}

}

// StudentController.java

**package** MVCPattern;

**public** **class** StudentController{

**private** Student model;

**private** StudentView view;

**public** StudentController(Student model, StudentView view){

**this**.model = model;

**this**.view = view;

}

**public** **void** updateGrade(**double** grade){

model.setGrade(grade);

}

**public** **void** showStudent(){

view.displayStudentDetails(model.getId(), model.getName(), model.getGrade());

}

}

// MVCTest.java

**package** MVCPattern;

**public** **class** MVCTest{

**public** **static** **void** main(String[] args){

Student model = **new** Student(2, "Shree", 85.0);

StudentView view = **new** StudentView();

StudentController controller = **new** StudentController(model, view);

controller.showStudent();

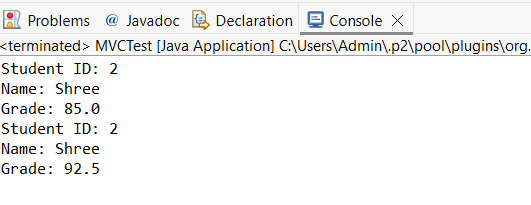
controller.updateGrade(92.5);

controller.showStudent();

}

}

**Output:**



**Exercise 11: Implementing Dependency Injection**

**Code:**

// CustomerRepository.java

**package** DependencyInjection;

**public** **interface** CustomerRepository{

String findCustomerById(**int** id);

}

// CustomerRepositoryImpl.java

**package** DependencyInjection;

**public** **class** CustomerRepositoryImpl **implements** CustomerRepository{

**public** String findCustomerById(**int** id){

**return** "Customer#" + id;

}

}

// CustomerService.java

**package** DependencyInjection;

**public** **class** CustomerService{

**private** CustomerRepository repo;

**public** CustomerService(CustomerRepository repo){

**this**.repo = repo;

}

**public** **void** printCustomer(**int** id) {

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

}

}

// DIApplication.java

**package** DependencyInjection;

**public** **class** DIApplication{

**public** **static** **void** main(String[] args){

CustomerRepository repo = **new** CustomerRepositoryImpl();

CustomerService service = **new** CustomerService(repo);

service.printCustomer(42);

}

}

**Output:**

