**1) Implementing the Singleton Pattern**

**Code**:

Logger.java

package SingletonPatternExample;

class Logger{

  private static Logger instance = null;

  private Logger(){}

  public static Logger getInstance(){

    if(instance==null) instance = new Logger();

    return instance;

  }

}

LoggerTest.java

package SingletonPatternExample;

public class LoggerTest {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

        if (logger1 == logger2) {

            System.out.println("PASS: Both logger instances are the same.");

        } else {

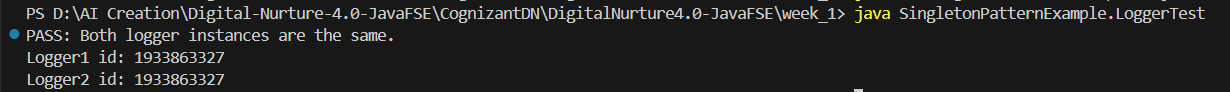
            System.out.println("FAIL: Logger instances are different.");

        }

    }

}

**Output:**



**2) Implementing the Factory Method Pattern**

**Code:**

Document.java

package FactoryMethodPatternExample;

public interface Document {

  public void open();

}

WordDocument.java

package FactoryMethodPatternExample;

public class WordDocument implements Document{

  public void open(){

      System.out.println("Openning Word Document");

    }

}

PdfDocument.java

package FactoryMethodPatternExample;

public class PdfDocument implements Document{

  public void open(){

      System.out.println("Openning Pdf Document");

    }

}

ExcelDocument.java

package FactoryMethodPatternExample;

public class ExcelDocument implements Document{

  public void open(){

      System.out.println("Openning Excel Document");

    }

}

DocumentFactory.java

package FactoryMethodPatternExample;

public abstract class DocumentFactory {

  public abstract Document createDocument();

}

WordDocumentFactory.java

package FactoryMethodPatternExample;

public class WordDocumentFactory extends DocumentFactory{

  public Document createDocument(){

    return new WordDocument();

  }

}

PdfDocumentFactory.java

package FactoryMethodPatternExample;

public class PdfDocumentFactory extends DocumentFactory {

  public Document createDocument() {

    return new PdfDocument();

  }

}

ExcelDocumentFactory.java

package FactoryMethodPatternExample;

public class ExcelDocumentFactory extends DocumentFactory {

  public Document createDocument() {

    return new ExcelDocument();

  }

}

DocumentFactoryTest.java

package FactoryMethodPatternExample;

public class DocumentFactoryTest {

    public static void main(String[] args) {

        DocumentFactory wordFactory = new WordDocumentFactory();

        Document wordDoc = wordFactory.createDocument();

        wordDoc.open();

        DocumentFactory pdfFactory = new PdfDocumentFactory();

        Document pdfDoc = pdfFactory.createDocument();

        pdfDoc.open();

        DocumentFactory excelFactory = new ExcelDocumentFactory();

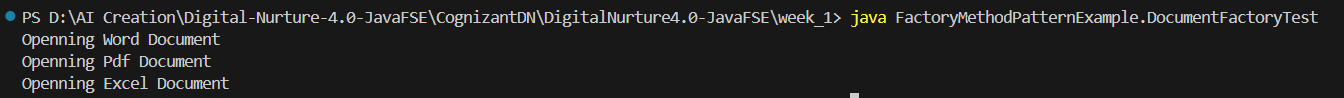
        Document excelDoc = excelFactory.createDocument();

        excelDoc.open();

    }

}

**Output:**



3) **Implementing the Builder Pattern**

**Code:**

Computer.java

package BuilderPatternExample;

public class Computer {

  private final String cpu;

  private final String ram;

  private final String os;

  private final String storage;

  private final String gpu;

  static class ComputerBuilder {

    private String cpu;

    private String ram;

    private String storage = "1TB HDD";

    private String os = "Ubuntu Linux";

    private String gpu = "None";

    ComputerBuilder(String cpu, String ram){

      this.cpu = cpu;

      this.ram = ram;

    }

    public ComputerBuilder setStorage(String Text){

      this.storage=Text;

      return this;

    }

    public ComputerBuilder setOs(String Text){

      this.os=Text;

      return this;

    }

    public ComputerBuilder setGpu(String Text){

      this.gpu=Text;

      return this;

    }

    public Computer build(){

      return new Computer(this);

    }

  }

  private Computer(ComputerBuilder ComputerBuild){

    this.cpu = ComputerBuild.cpu;

    this.ram = ComputerBuild.ram;

    this.os = ComputerBuild.os;

    this.storage = ComputerBuild.storage;

    this.gpu = ComputerBuild.gpu;

  }

  public String getCpu(){

    return this.cpu;

  }

  public String getRam(){

    return this.ram;

  }

  public String getOs(){

    return this.os;

  }

  public String getStorage(){

    return this.storage;

  }

  public String getGpu(){

    return this.gpu;

  }

}

ComputerTest.java

package BuilderPatternExample;

public class ComputerTest {

  public static void main(String[] args) {

      Computer myPC = new Computer.ComputerBuilder("Intel i7", "16GB")

                          .setStorage("1TB SSD")

                          .setGpu("NVIDIA RTX 4060")

                          .setOs("Windows 11")

                          .build();

      System.out.println("CPU: " + myPC.getCpu());

      System.out.println("RAM: " + myPC.getRam());

      System.out.println("Storage: " + myPC.getStorage());

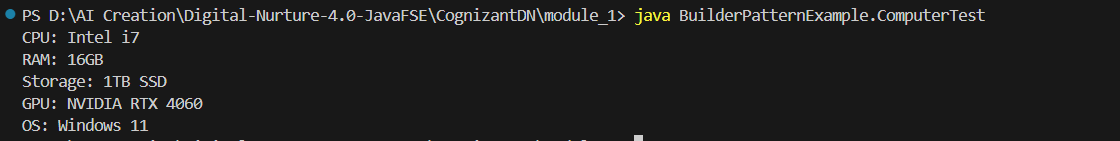
      System.out.println("GPU: " + myPC.getGpu());

      System.out.println("OS: " + myPC.getOs());

  }

}

**Output:**



**4)** **Implementing the Adapter Pattern**

**Code:**

PaymentProcessor.java

package AdapterPatternExample;

public interface PaymentProcessor {

  public void processPayment(double amount);

}

PaypalAdapter.java

package AdapterPatternExample;

public class PaypalAdapter implements PaymentProcessor {

  private PaypalGateway gateway;

  public PaypalAdapter(){

    gateway = new PaypalGateway();

  }

  public void processPayment(double amount) {

    gateway.makePayment(amount);

  }

}

StripeAdapter.java

package AdapterPatternExample;

public class StripeAdapter implements PaymentProcessor {

  private StripeGateway gateway;

  public StripeAdapter(){

    gateway = new StripeGateway();

  }

  public void processPayment(double amount) {

    gateway.Stripepay(amount);

  }

}

PaypalGateway.java

package AdapterPatternExample;

public class PaypalGateway {

  public void makePayment(double amount){

    System.err.println("Amount paid: Rs "+amount+" using PayPal");

  }

}

StripeGateway.java

package AdapterPatternExample;

public class StripeGateway {

  public void Stripepay(double amount){

    System.err.println("Amount paid: Rs "+amount+" using Stripe");

  }

}

PaymentProcessorTest.java

package AdapterPatternExample;

public class PaymentProcessorTest {

  public static void main(String[] args) {

    PaymentProcessor paypalProcessor = new PaypalAdapter();

    paypalProcessor.processPayment(2500);

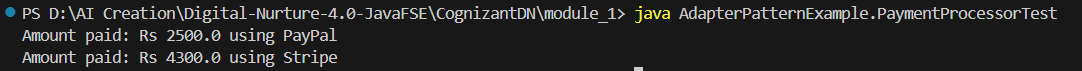
    PaymentProcessor stripeProcessor = new StripeAdapter();

    stripeProcessor.processPayment(4300);

  }

}

**Output:**



**5) Implementing the Decorator Pattern**

**Code:**

Notifier.java

package DecoratorPatternExample;

public interface Notifier {

  public void send(String Message);

}

EmailNotifier.java

package DecoratorPatternExample;

public class EmailNotifier implements Notifier{

  public void send(String Message) {

    System.out.println("Base Message: "+Message);

    System.out.println("Message sent via: Email");

  }

}

NotifierDecorator.java

package DecoratorPatternExample;

public abstract class NotifierDecorator implements Notifier{

  protected Notifier base;

  public NotifierDecorator(Notifier obj){

    this.base = obj;

  }

}

SlackNotifierDecorator.java

package DecoratorPatternExample;

public class SlackNotifierDecorator extends NotifierDecorator{

  public SlackNotifierDecorator(Notifier obj) {

    super(obj);

  }

  public void send(String Message) {

    base.send(Message);

    System.out.println("Message sent via: Slack");

  }

}

SMSNotifierDecorator.java

package DecoratorPatternExample;

public class SMSNotifierDecorator extends NotifierDecorator {

  public SMSNotifierDecorator(Notifier obj) {

    super(obj);

  }

  public void send(String Message) {

    base.send(Message);

    System.out.println("Message sent via: SMS");

  }

}

NotifierTest.java

package DecoratorPatternExample;

public class NotifierTest {

  public static void main(String[] args) {

    Notifier obj = new EmailNotifier();

    obj = new SMSNotifierDecorator(obj);

    obj.send("Hello");

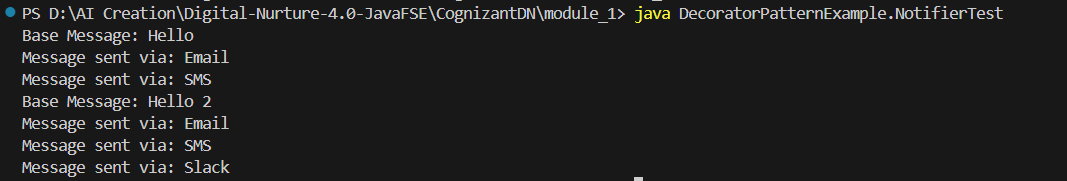
    obj = new SlackNotifierDecorator(obj);

    obj.send("Hello 2");

  }

}

**Output:**



**6) Implementing the Proxy Pattern**

**Code:**

Image.java

package ProxyPatternExample;

public interface Image {

  public void display();

}

RealImage.java

package ProxyPatternExample;

public class RealImage implements Image {

  public String path="NULL";

  RealImage(String path){

    this.path = path;

  }

  public void display() {

    System.out.println("Showing image: "+path);

  }

}

ProxyImage.java

package ProxyPatternExample;

public class ProxyImage implements Image{

  private RealImage realImage;

  private String path;

  public ProxyImage(String path){

    realImage = null;

    this.path = path;

    System.out.println("Only caching path: "+path);

  }

  public void display() {

    if(realImage == null) realImage=new RealImage(path);

    realImage.display();

  }

}

ImageTest.java

package ProxyPatternExample;

public class ImageTest {

  public static void main(String[] args) {

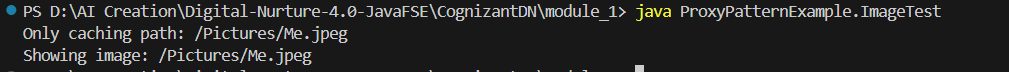
    Image img = new ProxyImage("/Pictures/Me.jpeg");

    img.display();

  }

}

**Output:**



**7) Implementing the Observer Pattern**

**Code:**

Stock.java

package ObserverPatternExample;

public interface Stock {

  public void register(Observer obj);

  public void deregister(Observer obj);

  public void notifyObservers();

}

StockMarket.java

package ObserverPatternExample;

import java.util.LinkedList;

public class StockMarket implements Stock{

  private LinkedList<Observer> observers;

  private final String name;

  private double price;

  public StockMarket(String stockName) {

    this.name = stockName;

    observers = new LinkedList<>();

  }

  public void setPrice(double price) {

    this.price = price;

    System.out.println("\n" + name + " price updated to: $" + price);

    notifyObservers();

  }

  public double getPrice() {

      return price;

  }

  public void register(Observer obj) {

    observers.add(obj);

  }

  public void deregister(Observer obj) {

    observers.remove(obj);

  }

  public void notifyObservers() {

    for (Observer o : observers) {

      o.update(name, price);

    }

  }

}

Observer.java

package ObserverPatternExample;

public interface Observer {

  public void update(String name, Double price);

}

WebApp.java

package ObserverPatternExample;

public class WebApp implements Observer{

  private String user;

  public WebApp(String user) {

      this.user = user;

  }

  public void update(String name, Double price) {

    System.out.println("[Web] " + user + " notified: " + name + " is now $" + price);

  }

}

MobileApp.java

package ObserverPatternExample;

public class MobileApp implements Observer{

  private String user;

  public MobileApp(String user) {

    this.user = user;

  }

  public void update(String name, Double price) {

    System.out.println("[Mobile] " + user + " notified: " + name + " is now $" + price);

  }

}

StockTest.java

package ObserverPatternExample;

public class StockTest {

  public static void main(String[] args) {

    StockMarket reliance = new StockMarket("Reliance");

    Observer mobileUser = new MobileApp("Alice");

    Observer webUser = new WebApp("Bob");

    reliance.register(mobileUser);

    reliance.register(webUser);

    reliance.setPrice(2500.50);

    reliance.setPrice(2515.75);

    // Unsubscribe Bob

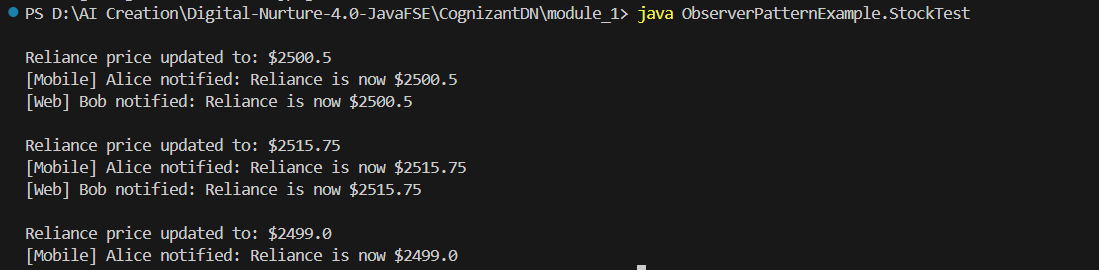
    reliance.deregister(webUser);

    reliance.setPrice(2499.00);

  }

}

**Output:**



**8) Implementing the Strategy Pattern**

**Code:**

PaymentStrategy.java

package StrategyPatternExample;

public interface PaymentStrategy {

  public void pay(double amount);

}

CreditCardPayment.java

package StrategyPatternExample;

public class CreditCardPayment implements PaymentStrategy{

  public void pay(double amount) {

    System.out.println("Paid $"+amount+" through CreditCard");

  }

}

PayPalPayment.java

package StrategyPatternExample;

public class PayPalPayment implements PaymentStrategy{

  public void pay(double amount) {

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

  }

}

PaymentContext.java

package StrategyPatternExample;

public class PaymentContext {

  private PaymentStrategy strategy;

  public PaymentContext(PaymentStrategy strategy){

    this.strategy = strategy;

  }

  public void changeContext(PaymentStrategy strategy){

    this.strategy = strategy;

  }

  public void pay(double amount){

    strategy.pay(amount);

  }

}

PaymentTest.java

package StrategyPatternExample;

public class PaymentTest {

  public static void main(String[] args) {

    PaymentStrategy card = new CreditCardPayment();

    PaymentStrategy online = new PayPalPayment();

    PaymentContext context = new PaymentContext(online);

    context.pay(1000.00);

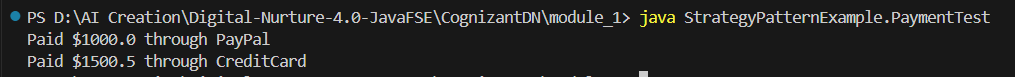
    context.changeContext(card);

    context.pay(1500.50);

  }

}

**Output:**



**9) Implementing the Command Pattern**

**Code:**

Command.java

package CommandPatternExample;

public interface Command {

  public void execute();

}

LightOffCommand.java

package CommandPatternExample;

public class LightOffCommand implements Command {

  private Light light;

  public LightOffCommand(Light light) {

    this.light = light;

  }

  public void execute() {

    light.turnOff();

  }

}

LightOnCommand.java

package CommandPatternExample;

public class LightOnCommand implements Command {

  private Light light;

  public LightOnCommand(Light light) {

    this.light = light;

  }

  public void execute() {

    light.turnOn();

  }

}

Light.java

package CommandPatternExample;

public class Light {

  public void turnOn() {

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

  }

  public void turnOff() {

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

  }

}

RemoteControl.java

package CommandPatternExample;

public class RemoteControl {

  private Command command;

  public void setCommand(Command command) {

    this.command = command;

  }

  public void pressButton() {

    if (command != null) {

      command.execute();

    } else {

      System.out.println("No command set.");

    }

  }

}

CommandTest.java

package CommandPatternExample;

public class CommandTest {

  public static void main(String[] args) {

    Light livingRoomLight = new Light();

    Command lightOn = new LightOnCommand(livingRoomLight);

    Command lightOff = new LightOffCommand(livingRoomLight);

    RemoteControl remote = new RemoteControl();

    remote.setCommand(lightOn);

    remote.pressButton();

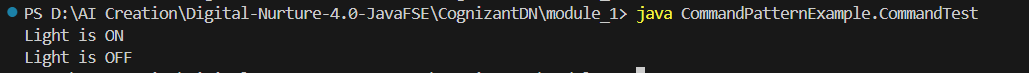
    remote.setCommand(lightOff);

    remote.pressButton();

  }

}

**Output:**



**10) Implementing the MVC Pattern**

**Code:**

Student.java

package MVCPatternExample;

public class Student {

  protected String id;

  protected String name;

  protected String grade;

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

    this.id = id;

    this.name = name;

    this.grade = grade;

  }

  public String getId() {

    return id;

  }

  public void setId(String id) {

    this.id = id;

  }

  public String getName() {

    return name;

  }

  public void setName(String name) {

    this.name = name;

  }

  public String getGrade() {

    return grade;

  }

  public void setGrade(String grade) {

    this.grade = grade;

  }

}

StudentView.java

package MVCPatternExample;

public class StudentView {

  public void displayStudentDetails(Student student) {

    System.out.println("Student Details:");

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

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

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

  }

}

StudentController.java

package MVCPatternExample;

// StudentController.java (Controller)

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 setStudentId(String id) {

    model.setId(id);

  }

  public void setStudentGrade(String grade) {

    model.setGrade(grade);

  }

  public String getStudentName() {

    return model.getName();

  }

  public String getStudentId() {

    return model.getId();

  }

  public String getStudentGrade() {

    return model.getGrade();

  }

  public void updateView() {

    view.displayStudentDetails(model);

  }

}

StudentTest.java

package MVCPatternExample;

public class StudentTest {

  public static void main(String[] args) {

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

    StudentView view = new StudentView();

    StudentController controller = new StudentController(student, view);

    controller.updateView();

    controller.setStudentName("Bob");

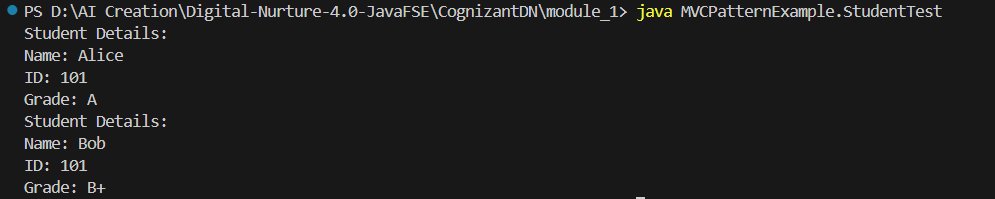
    controller.setStudentGrade("B+");

    controller.updateView();

  }

}

**Output:**



**11) Implementing Dependency Injection**

**Code:**

Customer.java

package DependencyInjectionExample;

public class Customer {

  private String id;

  private String name;

  public Customer(String id, String name) {

    this.id = id;

    this.name = name;

  }

  public String getId() {

    return id;

  }

  public String getName() {

    return name;

  }

}

CustomerRepository.java

package DependencyInjectionExample;

public interface CustomerRepository {

  void addCustomer(Customer customer);

  Customer findCustomerById(String id);

}

CustomerRepositoryImpl.java

package DependencyInjectionExample;

import java.util.HashMap;

public class CustomerRepositoryImpl implements CustomerRepository {

  HashMap<String, Customer> CustomerList;

  CustomerRepositoryImpl(){

    this.CustomerList = new HashMap<>();

  }

  public void addCustomer(Customer customer){

    CustomerList.put(customer.getId(), customer);

  }

  public Customer findCustomerById(String id) {

    return CustomerList.get(id);

  }

}

CustomerService.java

package DependencyInjectionExample;

public class CustomerService {

  private final CustomerRepository customerRepository;

  public CustomerService(CustomerRepository customerRepository) {

    this.customerRepository = customerRepository;

  }

  public void displayCustomer(String id) {

    Customer customer = customerRepository.findCustomerById(id);

    if (customer != null) {

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

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

    } else {

      System.out.println("Customer not found for ID: " + id);

    }

  }

}

CustomerTest.java

package DependencyInjectionExample;

public class CustomerTest {

  public static void main(String[] args) {

    CustomerRepository repository = new CustomerRepositoryImpl();

    repository.addCustomer(new Customer("001", "Tom"));

    repository.addCustomer(new Customer("002", "Bob"));

    CustomerService service = new CustomerService(repository);

    service.displayCustomer("001");

    service.displayCustomer("002");

  }

}

**Output:**

