1.SINGLETON PROGRAM

**package** sigleton;

**public** **class** singleton {

**static** **class** Logger {

**private** **static** Logger *l* = **new** Logger();

**private** Logger() {

System.***out***.println("Logger created");

}

**public** **static** Logger getLogger() {

**return** *l*;

}

**public** **void** logMessage(String msg) {

System.***out***.println("Log: " + msg);

}

}

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

{

Logger obj = Logger.*getLogger*();

obj.logMessage("Starting the application...");

Logger obj1 = Logger.*getLogger*();

obj1.logMessage("Continuing with the application...");

**if** (obj == obj1) {

System.***out***.println(" same logger");

} **else** {

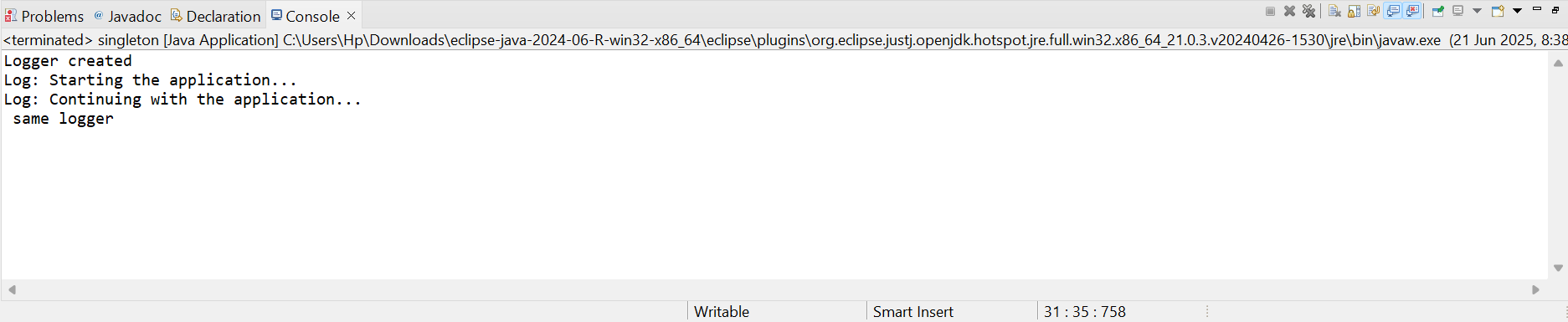
System.***out***.println(" different loggers");

}

}

}

OUTPUT:



2.FACTOR METHOD PATTERN

**public** **class** factormethodpatternexample {

**interface** MyDocument {

**void** openDocument();

}

**static** **class** WordDoc **implements** MyDocument {

**public** **void** openDocument() {

System.***out***.println("Word Document is now open.");

}

}

**static** **class** PdfDoc **implements** MyDocument {

**public** **void** openDocument() {

System.***out***.println("PDF Document is now open.");

}

}

**static** **class** ExcelDoc **implements** MyDocument {

**public** **void** openDocument() {

System.***out***.println("Excel Document is now open.");

}

}

**abstract** **static** **class** MyDocFactory {

**abstract** MyDocument createDocument();

}

**static** **class** WordDocFactory **extends** MyDocFactory {

**public** MyDocument createDocument() {

**return** **new** WordDoc();

}

}

**static** **class** PdfDocFactory **extends** MyDocFactory {

**public** MyDocument createDocument() {

**return** **new** PdfDoc();

}

}

**static** **class** ExcelDocFactory **extends** MyDocFactory {

**public** MyDocument createDocument() {

**return** **new** ExcelDoc();

}

}

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

{

MyDocFactory obj = **new** WordDocFactory();

MyDocument word = obj.createDocument();

word.openDocument();

MyDocFactory obj1 = **new** PdfDocFactory();

MyDocument pdf = obj1.createDocument();

pdf.openDocument();

MyDocFactory obj2 = **new** ExcelDocFactory();

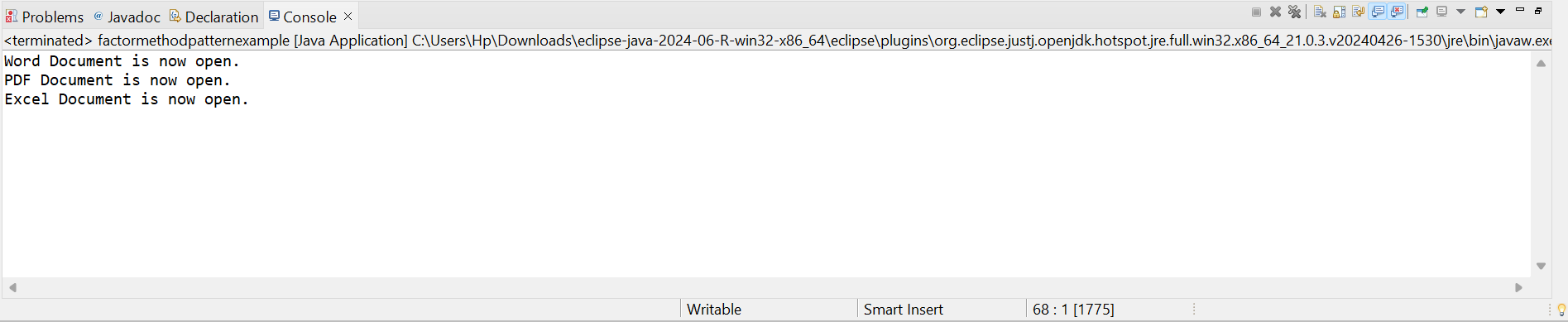
MyDocument excel = obj2.createDocument();

excel.openDocument();

}

}

OUTPUT:



3.BUILDER PATTERN

**package** builder;

**public** **class** mycomputerbuilderexample {

**static** **class** Computer {

String cpu;

String ram;

String storage;

String graphics;

**private** Computer(Builder b) {

cpu = b.cpu;

ram = b.ram;

storage = b.storage;

graphics = b.graphics;

}

**public** **void** showDetails() {

System.***out***.println("CPU: " + cpu);

System.***out***.println("RAM: " + ram);

System.***out***.println("Storage: " + storage);

System.***out***.println("Graphics: " + graphics);

}

**static** **class** Builder {

String cpu;

String ram;

String storage;

String graphics;

**public** Builder setCpu(String c) {

cpu = c;

**return** **this**;

}

**public** Builder setRam(String r) {

ram = r;

**return** **this**;

}

**public** Builder setStorage(String s) {

storage = s;

**return** **this**;

}

**public** Builder setGraphics(String g) {

graphics = g;

**return** **this**;

}

**public** Computer build() {

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

}

}

}

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

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

.setCpu("i3")

.setRam("4GB")

.setStorage("500GB HDD")

.build();

obj.showDetails();

System.***out***.println();

Computer obj1 = **new** Computer.Builder()

.setCpu("Ryzen 7")

.setRam("16GB")

.setStorage("1TB SSD")

.setGraphics("NVIDIA RTX 3060")

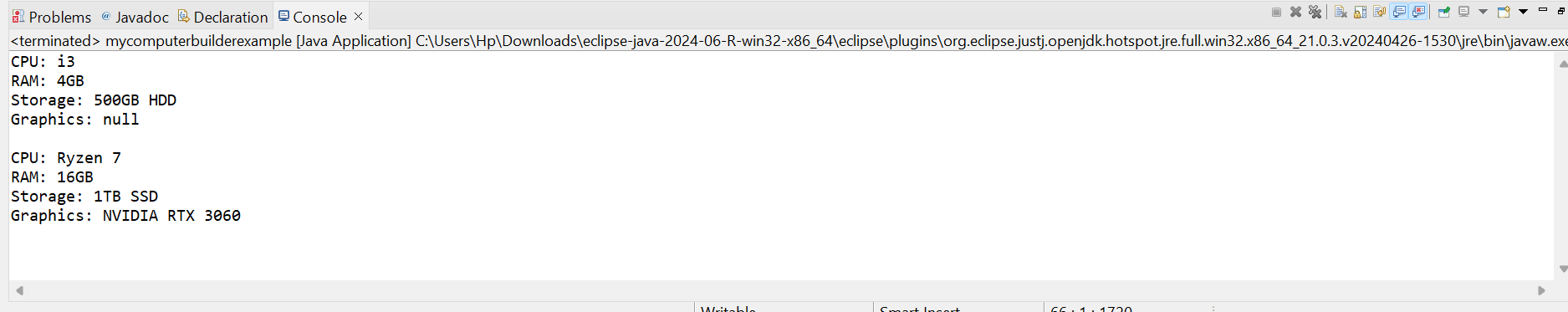
.build();

obj1.showDetails();

}

}

OUTPUT:



4.ADAPTER PATTERN

**package** adapter;

**public** **class** myadapterpatternexample {

**interface** PaymentProcessor {

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

}

**static** **class** PayPalGateway {

**public** **void** sendMoney(**double** money) {

System.***out***.println(money +" processed.");

}

}

**static** **class** StripeGateway {

**public** **void** makeStripePayment(**double** value) {

System.***out***.println(value + "processed.");

}

}

**static** **class** PayPalAdapter **implements** PaymentProcessor {

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

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

paypal.sendMoney(amount);

}

}

**static** **class** StripeAdapter **implements** PaymentProcessor {

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

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

stripe.makeStripePayment(amount);

}

}

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

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

obj.processPayment(250);

System.***out***.println();

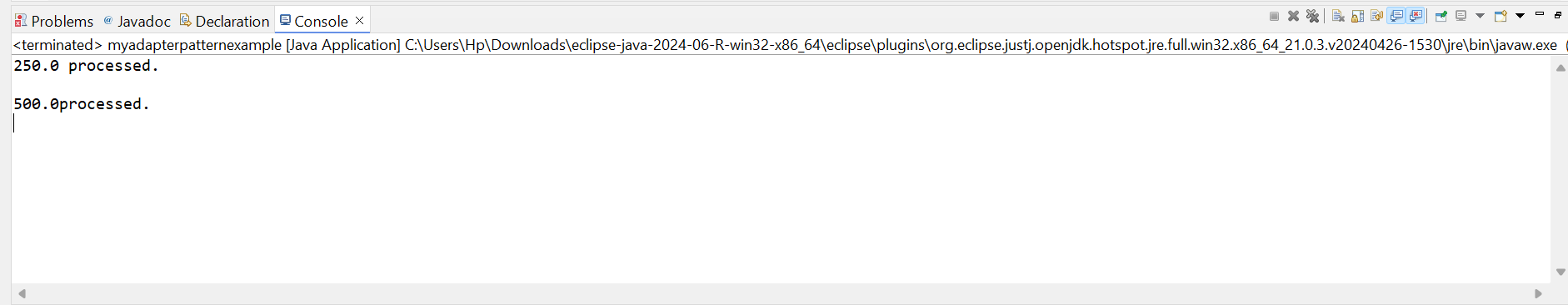
PaymentProcessor obj1 = **new** StripeAdapter();

obj1.processPayment(500);

}

}

OUTPUT:



5.DECORATOR PATTERN

**package** decorative;

**public** **class** mydecoratepatternexample {

**interface** Notifier {

**void** send(String message);

}

**static** **class** EmailNotifier **implements** Notifier {

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

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

}

}

**static** **abstract** **class** NotifierDecorator **implements** Notifier {

**protected** Notifier notifier;

**public** NotifierDecorator(Notifier notifier) {

**this**.notifier = notifier;

}

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

notifier.send(message);

}

}

**static** **class** SMSNotifierDecorator **extends** NotifierDecorator {

**public** SMSNotifierDecorator(Notifier notifier) {

**super**(notifier);

}

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

**super**.send(message);

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

}

}

**static** **class** SlackNotifierDecorator **extends** NotifierDecorator {

**public** SlackNotifierDecorator(Notifier notifier) {

**super**(notifier);

}

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

**super**.send(message);

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

}

}

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

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

System.***out***.println("Email Only:");

obj.send("Hello");

System.***out***.println();

Notifier obj1 = **new** SMSNotifierDecorator(**new** EmailNotifier());

obj1.send("Order placed");

System.***out***.println();

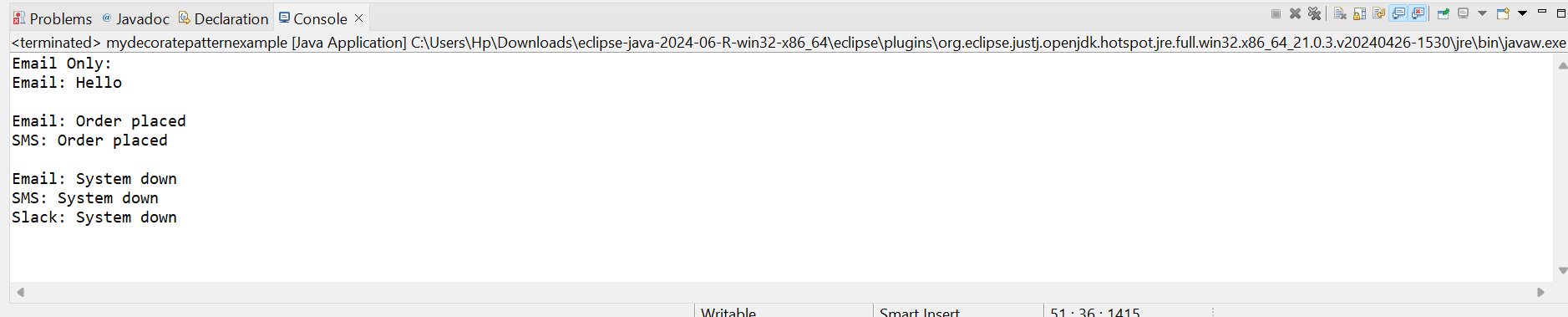
Notifier obj2= **new** SlackNotifierDecorator(**new** SMSNotifierDecorator(**new** EmailNotifier()));

obj2.send("System down");

}

}

OUTPUT:



6.PROXY PATTERN

**package** proxy;

**public** **class** myproxypatternexample {

**interface** Image {

**void** display();

}

**static** **class** RealImage **implements** Image {

**private** String fileName;

**public** RealImage(String fileName) {

**this**.fileName = fileName;

loadFromRemoteServer(fileName);

}

**private** **void** loadFromRemoteServer(String fileName) {

System.***out***.println("Loading image from server: " + fileName);

}

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

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

}

}

**static** **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();

}

}

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

Image img1 = **new** ProxyImage("nature.jpg");

System.***out***.println("First time display:");

img1.display();

System.***out***.println();

System.***out***.println("Second time display:");

img1.display();

System.***out***.println();

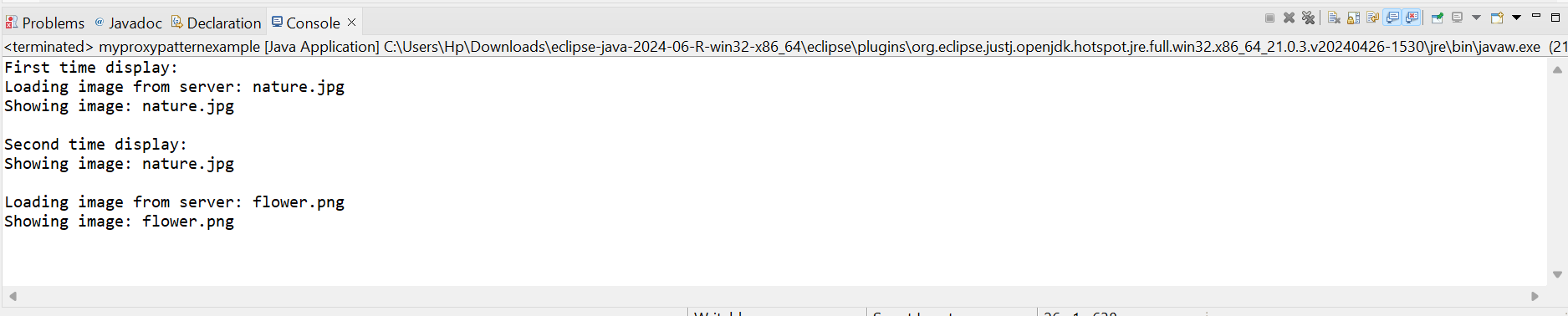
Image img2 = **new** ProxyImage("flower.png");

img2.display();

}

}

OUTPUT:



7.OBSERVER PATTERN

**package** observer;

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

**public** **class** myobserverpatternexample

{

**interface** Observer {

**void** update(String stock, **double** price);

}

**interface** Stock {

**void** register(Observer o);

**void** remove(Observer o);

**void** notifyAllObservers();

}

**static** **class** StockMarket **implements** Stock {

List<Observer> list = **new** ArrayList<>();

String stockName;

**double** stockPrice;

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

stockName = stock;

stockPrice = price;

notifyAllObservers();

}

**public** **void** register(Observer o) {

list.add(o);

}

**public** **void** remove(Observer o) {

list.remove(o);

}

**public** **void** notifyAllObservers() {

**for** (Observer o : list) {

o.update(stockName, stockPrice);

}

}

}

**static** **class** MobileApp **implements** Observer {

String name;

MobileApp(String name) {

**this**.name = name;

}

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

System.***out***.println(name + " Mobile: " + stock + " = " + price);

}

}

**static** **class** WebApp **implements** Observer {

String name;

WebApp(String name) {

**this**.name = name;

}

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

System.***out***.println(name + " Web: " + stock + " = " + price);

}

}

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

StockMarket sm = **new** StockMarket();

Observer mob = **new** MobileApp("User1");

Observer web = **new** WebApp("User2");

sm.register(mob);

sm.register(web);

sm.setPrice("TCS", 3200);

System.***out***.println();

sm.setPrice("INFY", 1500);

System.***out***.println();

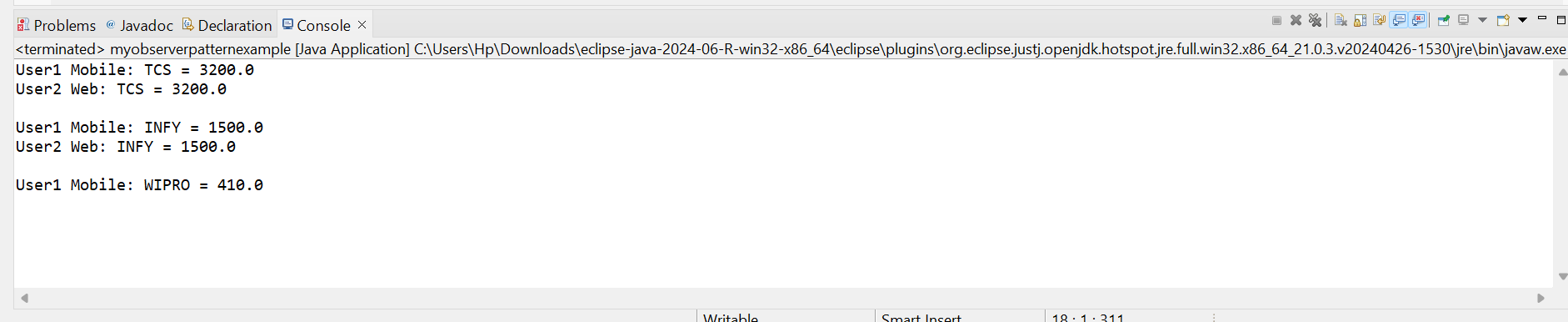
sm.remove(web);

sm.setPrice("WIPRO", 410);

}

}

OUTPUT:



8.STRATEGY PATTERN

**public** **class** MyStrategyPatternExample {

**interface** PaymentStrategy {

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

}

**static** **class** CreditCardPayment **implements** PaymentStrategy {

**private** String cardNumber;

**public** CreditCardPayment(String cardNumber) {

**this**.cardNumber = cardNumber;

}

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

System.***out***.println("Paid " + amount + " using Credit Card: " + cardNumber);

}

}

**static** **class** PayPalPayment **implements** PaymentStrategy {

**private** String email;

**public** PayPalPayment(String email) {

**this**.email = email;

}

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

System.***out***.println("Paid " + amount + " using PayPal: " + email);

}

}

**static** **class** PaymentContext {

**private** PaymentStrategy strategy;

**public** **void** setPaymentStrategy(PaymentStrategy strategy) {

**this**.strategy = strategy;

}

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

**if** (strategy != **null**) {

strategy.pay(amount);

} **else** {

System.***out***.println("No payment method selected.");

}

}

}

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

PaymentContext obj = **new** PaymentContext();

obj.setPaymentStrategy(**new** CreditCardPayment("1234-5678-9012-3456"));

obj.makePayment(2500);

System.***out***.println();

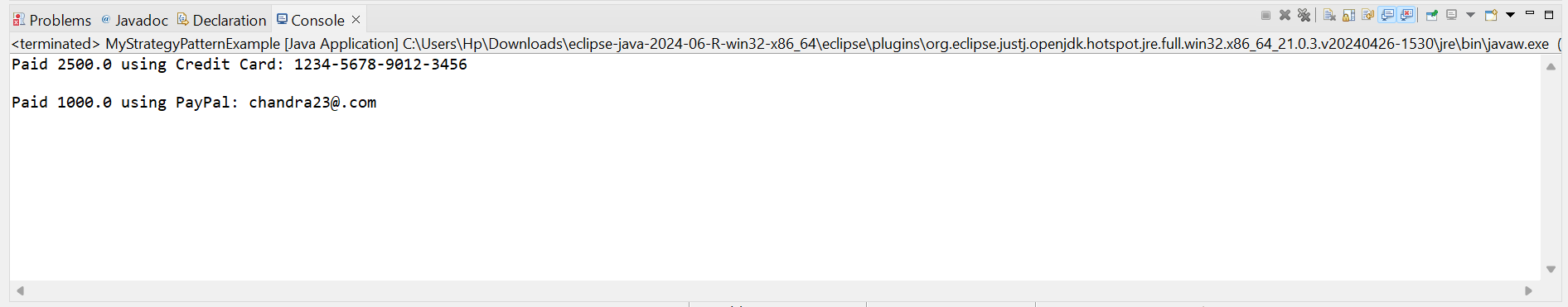
obj.setPaymentStrategy(**new** PayPalPayment("chandra23@.com"));

obj.makePayment(1000);

}

}

OUTPUT:



9.COMMAND PATTERN

**package** command;

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

**public** **class** mycommandpatternexample {

**interface** Command {

**void** execute();

}

**static** **class** Light {

**public** **void** turnOn() {

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

}

**public** **void** turnOff() {

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

}

}

**static** **class** LightOnCommand **implements** Command {

**private** Light light;

**public** LightOnCommand(Light light) {

**this**.light = light;

}

**public** **void** execute() {

light.turnOn();

}

}

**static** **class** LightOffCommand **implements** Command {

**private** Light light;

**public** LightOffCommand(Light light) {

**this**.light = light;

}

**public** **void** execute() {

light.turnOff();

}

}

**static** **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");

}

}

}

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

Light obj = **new** Light();

Command on = **new** LightOnCommand(obj);

Command off = **new** LightOffCommand(obj);

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

remote.setCommand(on);

remote.pressButton();

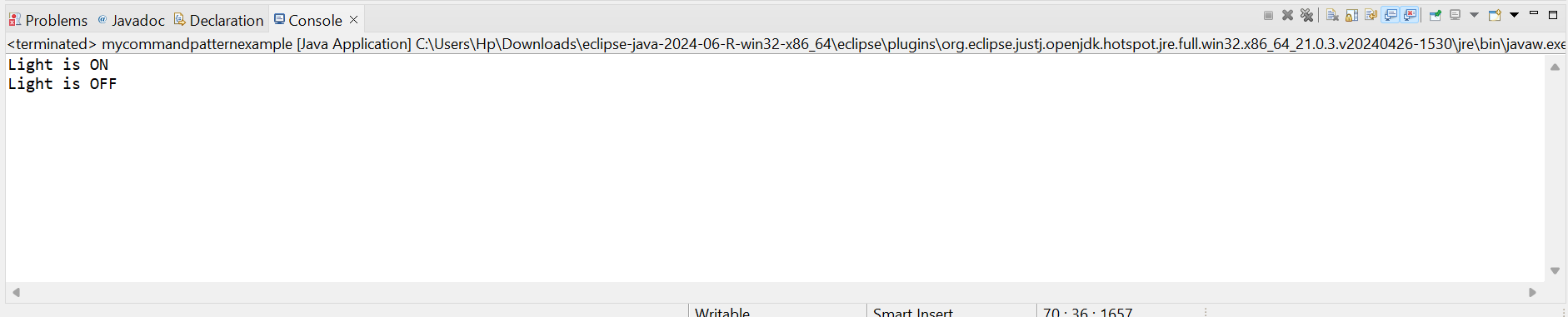
remote.setCommand(off);

remote.pressButton();

}

}

OUTPUT:



10.MVC PATTERN

**package** mvc;

**public** **class** mymvc {

**static** **class** Student {

String name;

String id;

String grade;

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

**this**.name = name;

**this**.id = id;

**this**.grade = grade;

}

}

**static** **class** StudentView {

**void** show(String name, String id, String grade) {

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

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

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

}

}

**static** **class** StudentController {

Student s;

StudentView v;

StudentController(Student s, StudentView v) {

**this**.s = s;

**this**.v = v;

}

**void** setName(String name) {

s.name = name;

}

**void** setGrade(String grade) {

s.grade = grade;

}

**void** showDetails() {

v.show(s.name, s.id, s.grade);

}

}

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

Student s1 = **new** Student("John", "S001", "B");

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

StudentController ctrl = **new** StudentController(s1, view);

ctrl.showDetails();

System.***out***.println();

ctrl.setName("Alice");

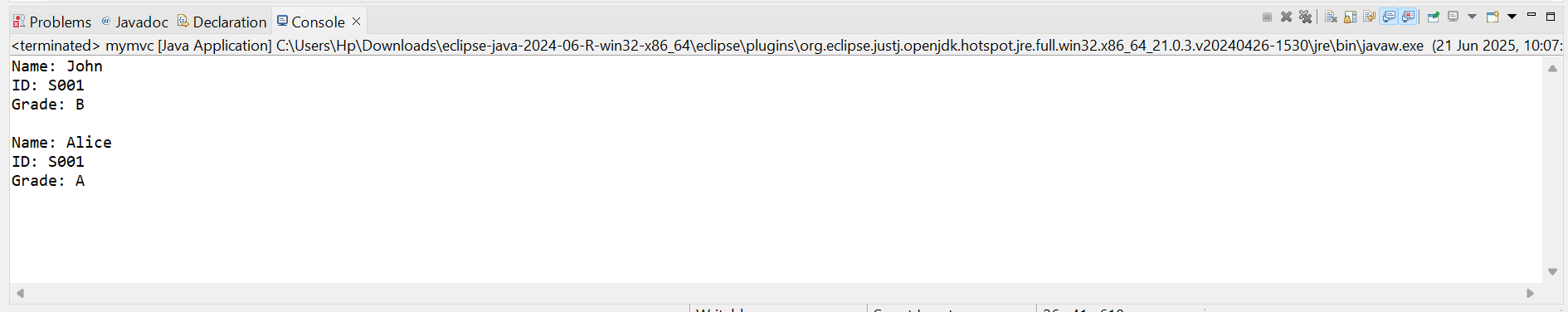
ctrl.setGrade("A");

ctrl.showDetails();

}

}

OUPUT:



11.DEPENDENCY INJECTION

**package** dependency;

**public** **class** mydependencyinjection {

**interface** CustomerRepository {

String findCustomerById(String id);

}

**static** **class** CustomerRepositoryImpl **implements** CustomerRepository {

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

**return** "Customer ID: " + id + ", Name: chandra";

}

**static** **class** CustomerService {

CustomerRepository repo;

CustomerService(CustomerRepository repo) {

**this**.repo = repo;

}

**void** showCustomer(String id) {

String customer = repo.findCustomerById(id);

System.***out***.println(customer);

}

}

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

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

CustomerService obj = **new** CustomerService(cr);

obj.showCustomer("22004");

}

}

OUTPUT:

