**Week 1**

Mandatory

Design patterns and principles

1. Singleton pattern

Logger.java (class)

**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: " + message);**

**}**

**}**

Main.java(class)

**public class Main {**

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

**Logger logger1 = Logger.*getInstance*();**

**Logger logger2 = Logger.*getInstance*();**

**logger1.log("This is the first log message.");**

**logger2.log("This is the second log message.");**

**if (logger1 == logger2) {**

**System.*out*.println("Both logger instances are the same (Singleton works).");**

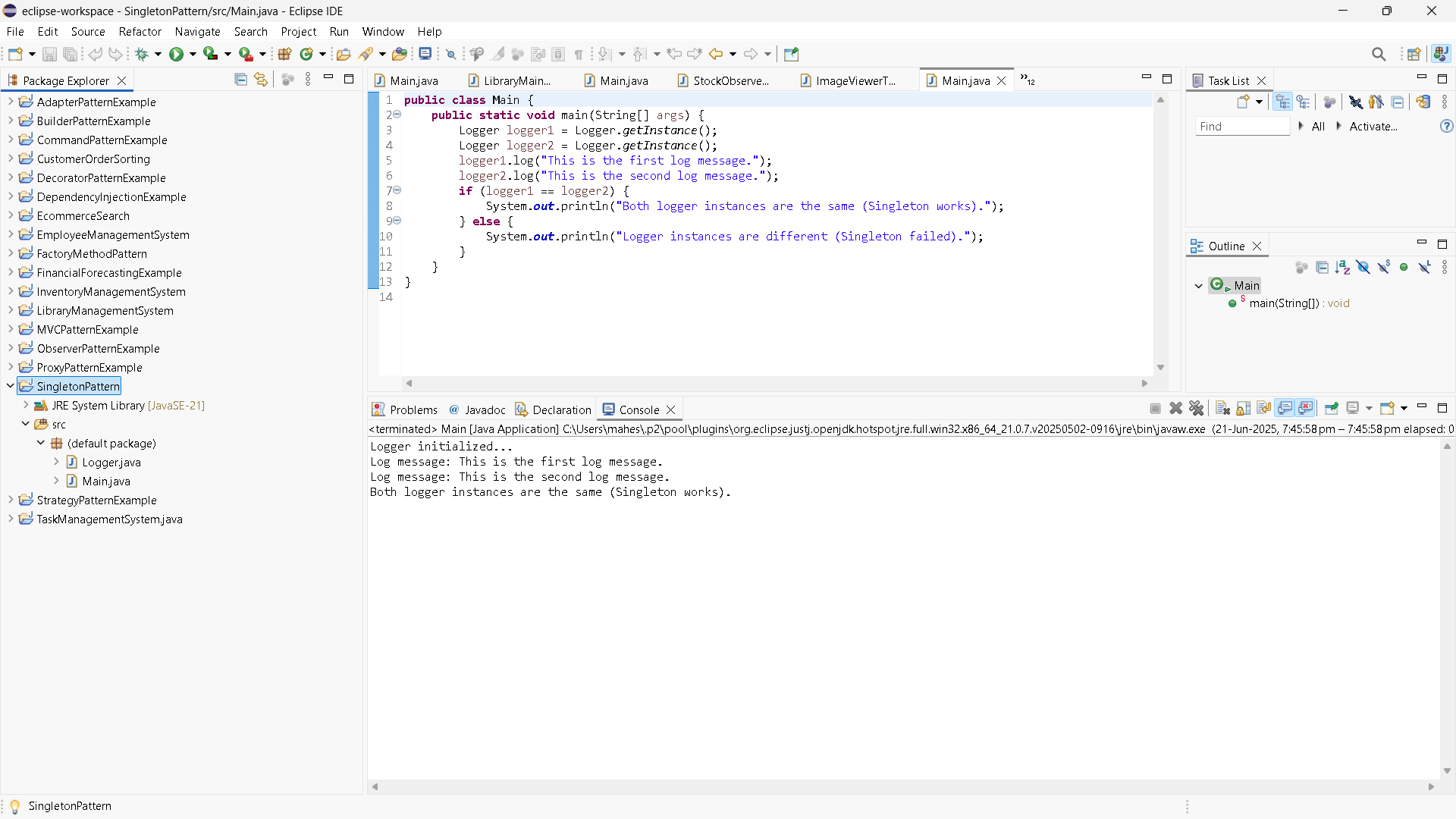
**} else {**

**System.*out*.println("Logger instances are different (Singleton failed).");**

**}**

**}**

**}**



1. Factory method Pattern

Document.java (interface)

**public** **interface** Document {

**void** open();

}

DocumentFactory.java

**public** **abstract** **class** DocumentFactory {

**public** **abstract** Document createDocument();

}

ExcelDocument.java

**public class ExcelDocument implements Document {**

**@Override**

**public void open() {**

**System.*out*.println("Opening an Excel document.");**

**}**

**}**

ExcelDocumentFactory.java

**public class ExcelDocumentFactory extends DocumentFactory {**

**@Override**

**public Document createDocument() {**

**return new ExcelDocument();**

**}**

**}**

PdfDocument.java

**public class PdfDocument implements Document {**

**@Override**

**public void open() {**

**System.*out*.println("Opening a PDF document.");**

**}**

**}**

PdfDocumentFactory.java

**public class PdfDocumentFactory extends DocumentFactory {**

**@Override**

**public Document createDocument() {**

**return new PdfDocument();**

**}**

**}**

WordDocument.java

**public class WordDocument implements Document {**

**@Override**

**public void open() {**

**System.*out*.println("Opening a Word document.");**

**}**

**}**

WordDocumentFactory.java

**public class WordDocumentFactory extends DocumentFactory {**

**@Override**

**public Document createDocument() {**

**return new WordDocument();**

**}**

**}**

Main.java

**public class Main {**

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

**// Word document**

**DocumentFactory wordFactory = new WordDocumentFactory();**

**Document wordDoc = wordFactory.createDocument();**

**wordDoc.open();**

**// PDF document**

**DocumentFactory pdfFactory = new PdfDocumentFactory();**

**Document pdfDoc = pdfFactory.createDocument();**

**pdfDoc.open();**

**// Excel document**

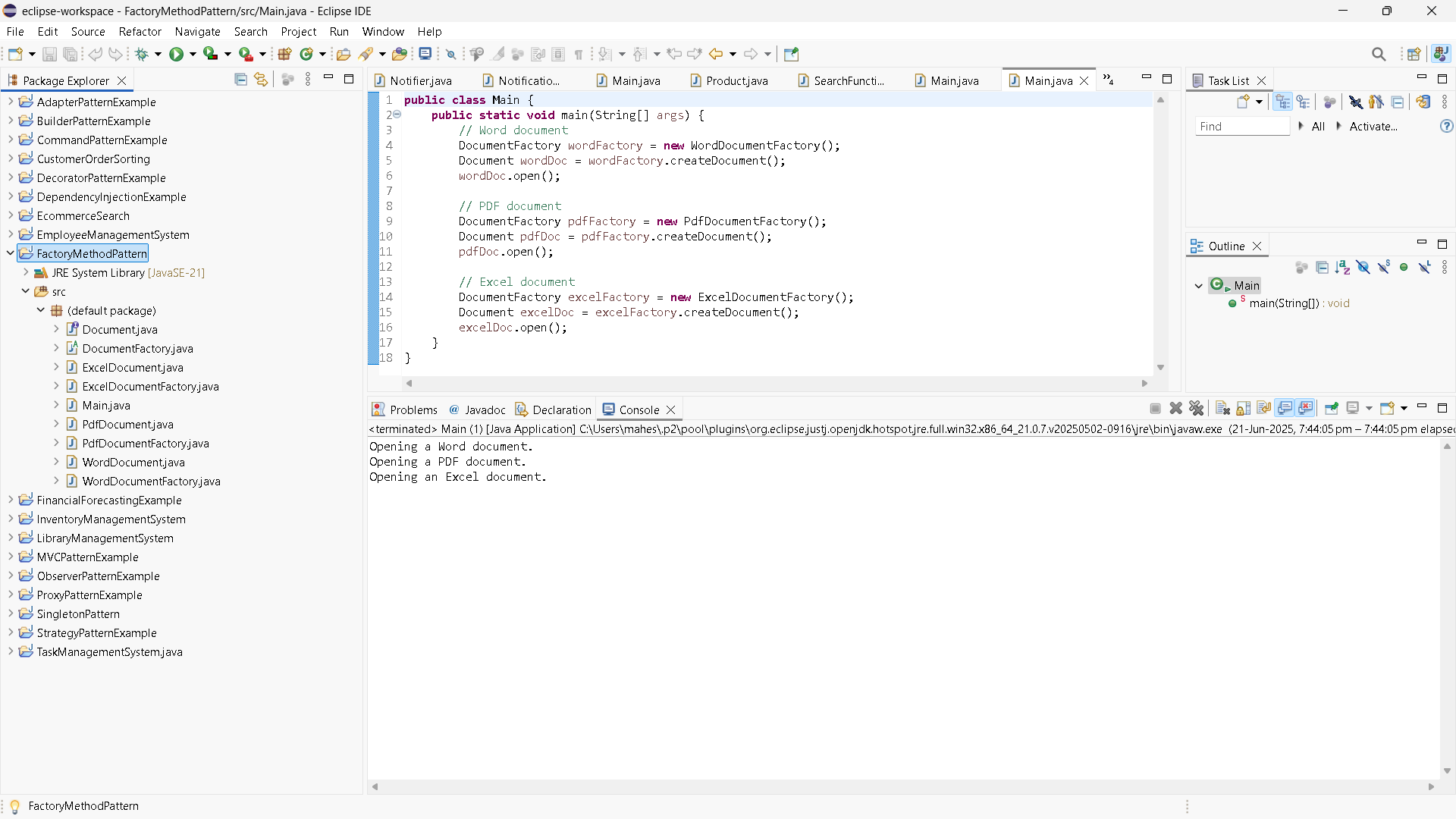
**DocumentFactory excelFactory = new ExcelDocumentFactory();**

**Document excelDoc = excelFactory.createDocument();**

**excelDoc.open();**

**}**

**}**



Practice problems

1. BuilderPAtternExample

Computer.java

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

// Required

**private** String CPU;

**private** String RAM;

// Optional

**private** String storage;

**private** String graphicsCard;

**private** String keyboard;

**private** String monitor;

// Private constructor (Builder)

**private** Computer(Builder builder) {

**this**.CPU = builder.CPU;

**this**.RAM = builder.RAM;

**this**.storage = builder.storage;

**this**.graphicsCard = builder.graphicsCard;

**this**.keyboard = builder.keyboard;

**this**.monitor = builder.monitor;

}

// Static nested Builder class

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

**private** String CPU;

**private** String RAM;

**private** String storage;

**private** String graphicsCard;

**private** String keyboard;

**private** String monitor;

**public** Builder(String CPU, String RAM) {

**this**.CPU = CPU;

**this**.RAM = RAM;

}

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

**this**.storage = storage;

**return** **this**;

}

**public** Builder setGraphicsCard(String graphicsCard) {

**this**.graphicsCard = graphicsCard;

**return** **this**;

}

**public** Builder setKeyboard(String keyboard) {

**this**.keyboard = keyboard;

**return** **this**;

}

**public** Builder setMonitor(String monitor) {

**this**.monitor = monitor;

**return** **this**;

}

**public** Computer build() {

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

}

}

// display

**public** **void** showSpecs() {

System.***out***.println("Computer Configuration:");

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

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

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

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

System.***out***.println("Keyboard: " + keyboard);

System.***out***.println("Monitor: " + monitor);

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

}

}

Main.java

**public** **class** Main {

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

// High-end Gaming PC

Computer gamingPC = **new** Computer.Builder("Intel i9", "32GB")

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 4090")

.setKeyboard("Mechanical RGB")

.setMonitor("4K 144Hz")

.build();

// Budget Office PC

Computer officePC = **new** Computer.Builder("Intel i3", "8GB")

.setStorage("500GB HDD")

.setMonitor("1080p")

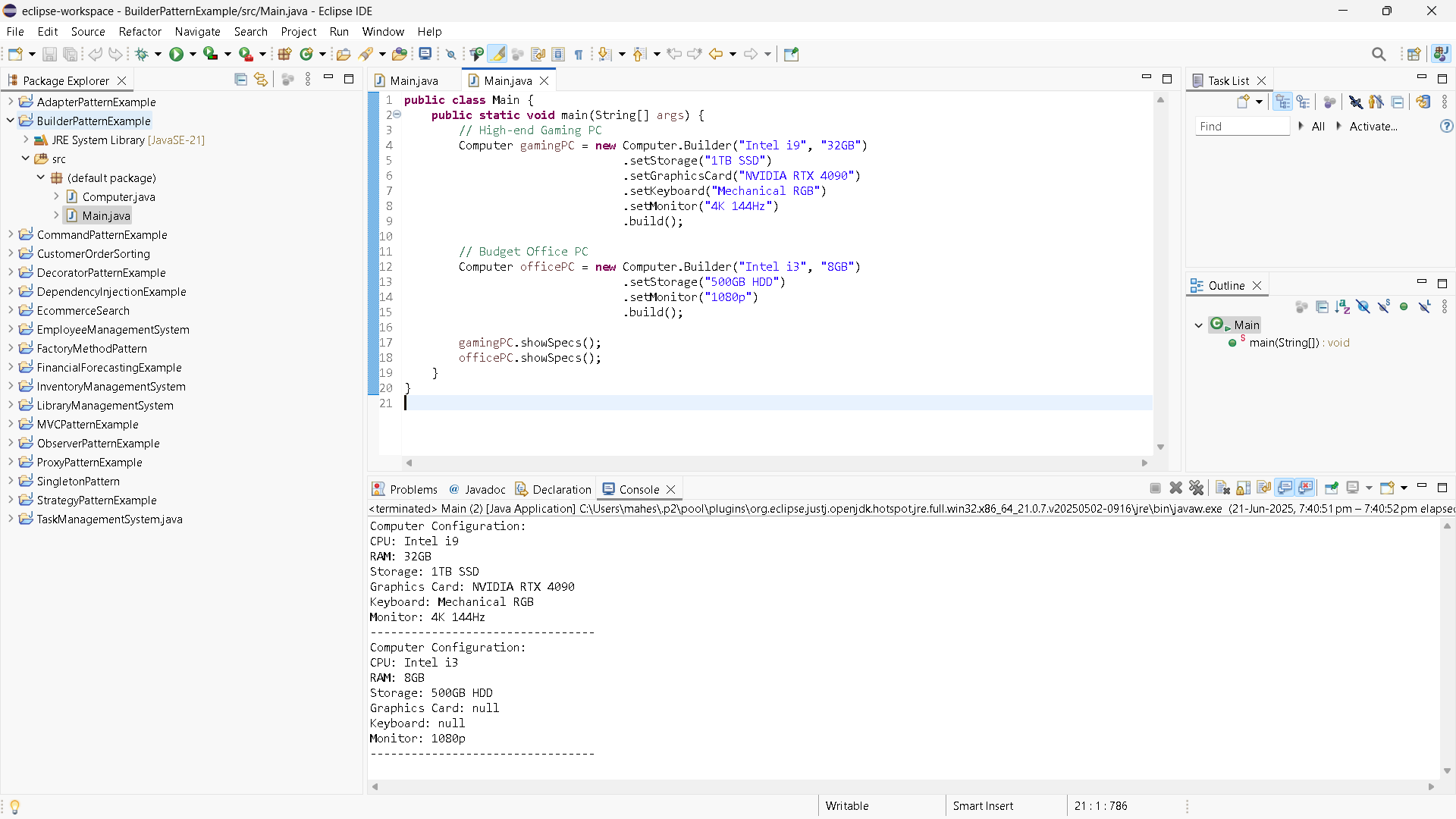
.build();

gamingPC.showSpecs();

officePC.showSpecs();

}

}



1. AdapterPatternExample

**Main.java**

**public** **class** Main {

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

PaymentProcessor paypalProcessor = **new** PayPalAdapter(**new** PayPalGateway());

PaymentProcessor stripeProcessor = **new** StripeAdapter(**new** StripeGateway());

PaymentProcessor razorpayProcessor = **new** RazorpayAdapter(**new** RazorpayGateway());

paypalProcessor.processPayment(1050.0);

stripeProcessor.processPayment(2500.32);

razorpayProcessor.processPayment(959.99);

}

}

PaymentProcessor.java

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

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

}

PayPalAdapter.java

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

**private** PayPalGateway payPal;

**public** PayPalAdapter(PayPalGateway payPal) {

**this**.payPal = payPal;

}

@Override

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

payPal.sendPayment(amount);

}

}

PayPalGateway.java

**public** **class** PayPalGateway {

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

System.***out***.println("Payment of Rs. " + amount + " processed via PayPal.");

}

}

RazorpayAdapter.java

**public** **class** RazorpayAdapter **implements** PaymentProcessor {

**private** RazorpayGateway razorpay;

**public** RazorpayAdapter(RazorpayGateway razorpay) {

**this**.razorpay = razorpay;

}

@Override

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

razorpay.doTransaction(amount);

}

}

RazorpayGateway.java

**public** **class** RazorpayGateway {

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

System.***out***.println("Payment of Rs. " + amount + " processed via Razorpay.");

}

}

StripeAdapter.java

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

**private** StripeGateway stripe;

**public** StripeAdapter(StripeGateway stripe) {

**this**.stripe = stripe;

}

@Override

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

stripe.makePayment(amount);

}

}

StripeGateway.java

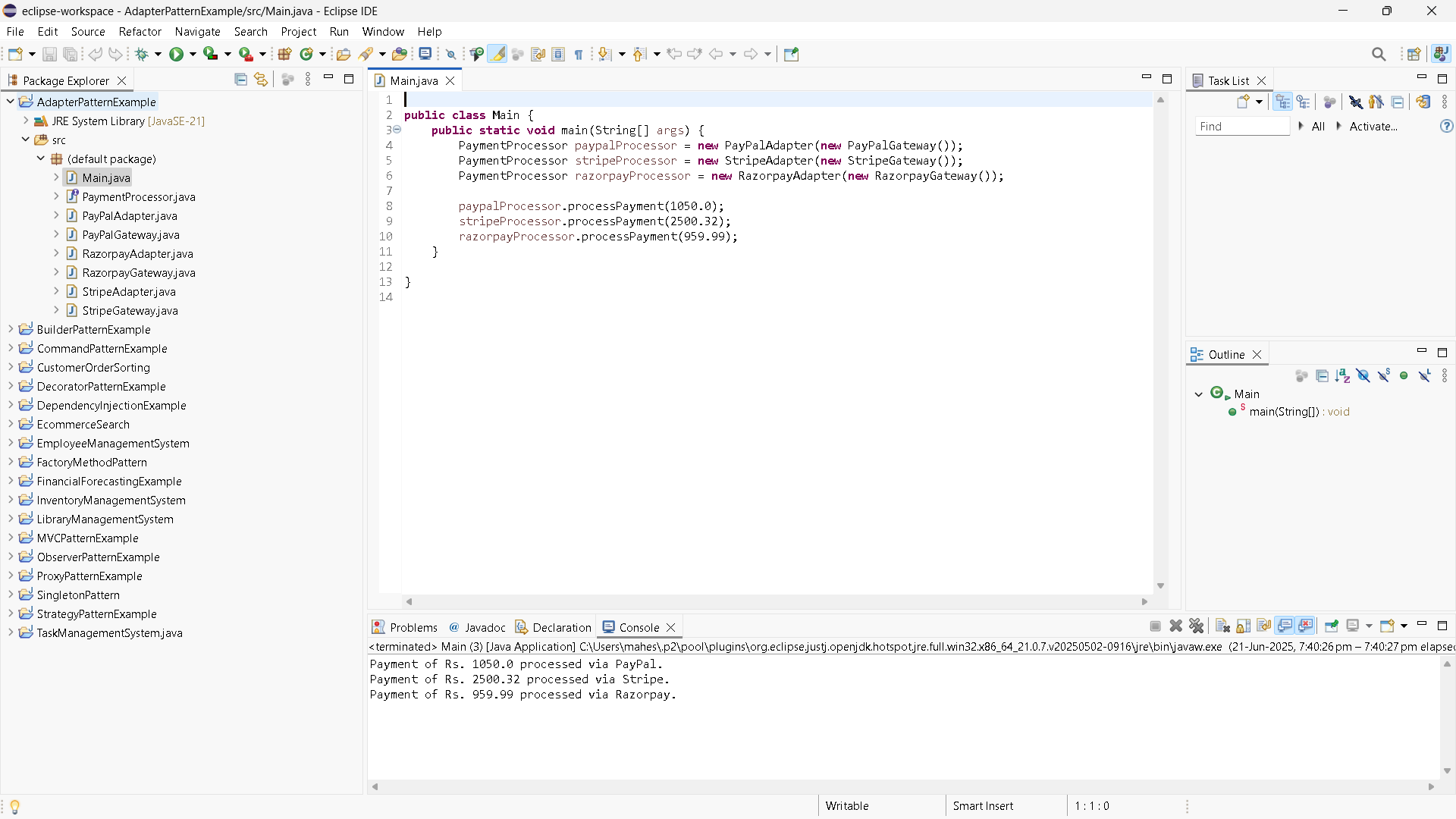
**public** **class** StripeGateway {

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

System.***out***.println("Payment of Rs. " + amount + " processed via Stripe.");

}

}



1. DecoratorPatternExample

EmailNotifier.java

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

@Override

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

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

}

}

NotificationTest.java

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

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

System.***out***.println("---Single Channel Notification ---");

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

emailOnly.send("System maintenance at 8 PM.");

System.***out***.println("\n---Email + SMS Notification ---");

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

emailSms.send("New login detected.");

System.***out***.println("\n--- Email + SMS + Slack Notification ---");

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

fullNotifier.send("Weekly report available.");

}

}

Notifier.java

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

**void** send(String message);

}

NotifierDecorator.java

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

**protected** Notifier notifier;

**public** NotifierDecorator(Notifier notifier) {

**this**.notifier = notifier;

}

@Override

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

notifier.send(message); // delegate

}

}

SlackNotifierDecorator.java

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

**public** SlackNotifierDecorator(Notifier notifier) {

**super**(notifier);

}

@Override

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

**super**.send(message); // previous

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

}

}

SMSNotifierDecorator.java

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

**public** SMSNotifierDecorator(Notifier notifier) {

**super**(notifier);

}

@Override

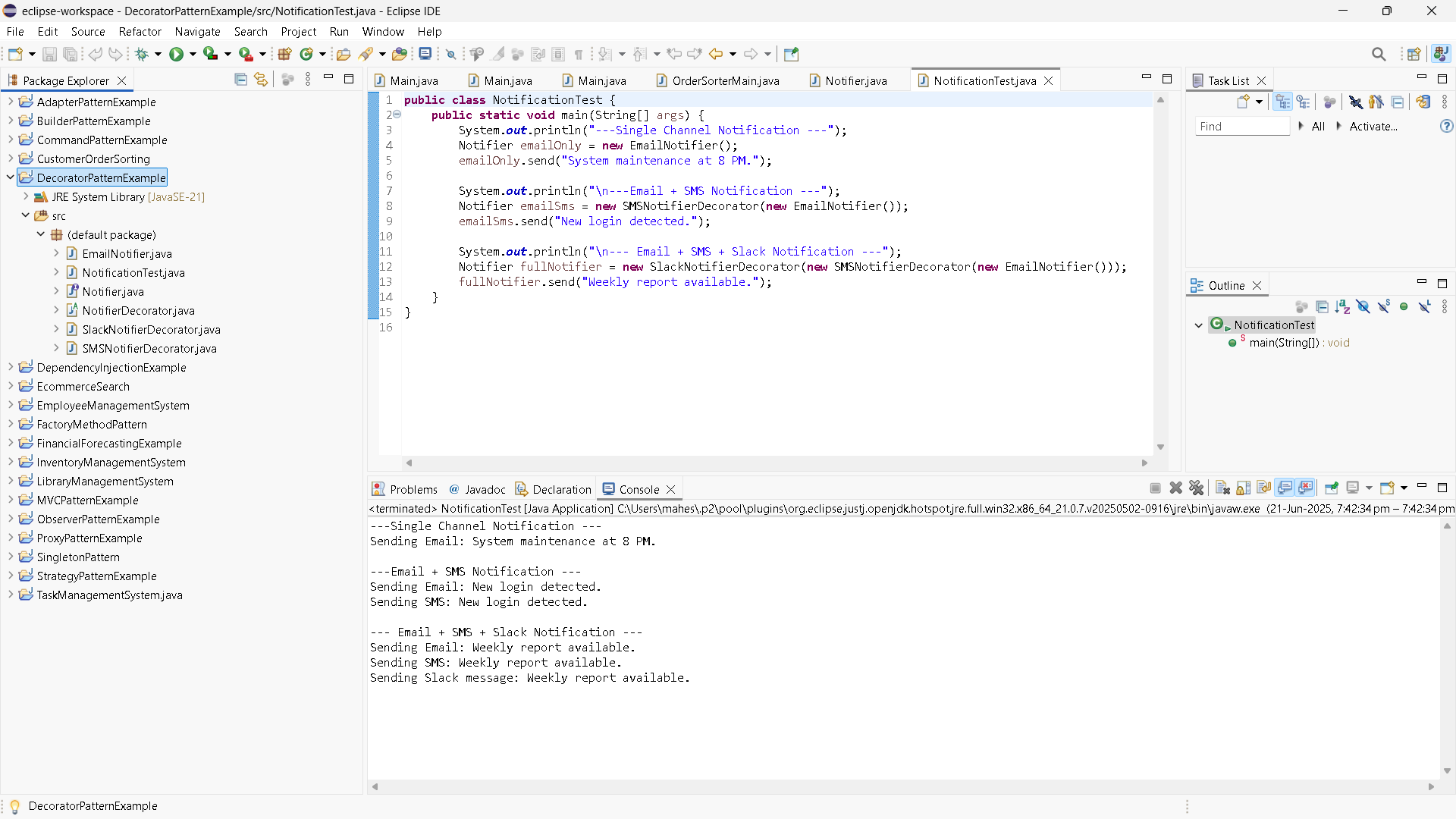
**public** **void** send(String message) {

**super**.send(message); // Email or previous

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

}

}



1. ProxyPatternExample

Image.java

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

**void** display();

}

ImageViewerTest.java

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

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

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

Image img2 = **new** ProxyImage("portrait.jpg");

System.***out***.println("\n-First time display (lazy load) -");

img1.display(); // Loads and displays

System.***out***.println("\n- Display again (uses cache) -");

img1.display(); // Uses cache

System.***out***.println("\n- Display another image (lazy load) -");

img2.display(); // Loads and displays

}

}

ProxyImage.java

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

**private** String fileName;

**private** RealImage realImage;

**public** ProxyImage(String fileName) {

**this**.fileName = fileName;

}

@Override

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

**if** (realImage == **null**) {

realImage = **new** RealImage(fileName); // lazy initialization

} **else** {

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

}

realImage.display();

}

}

RealImage.java

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

**private** String fileName;

**public** RealImage(String fileName) {

**this**.fileName = fileName;

loadFromServer(fileName);

}

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

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

}

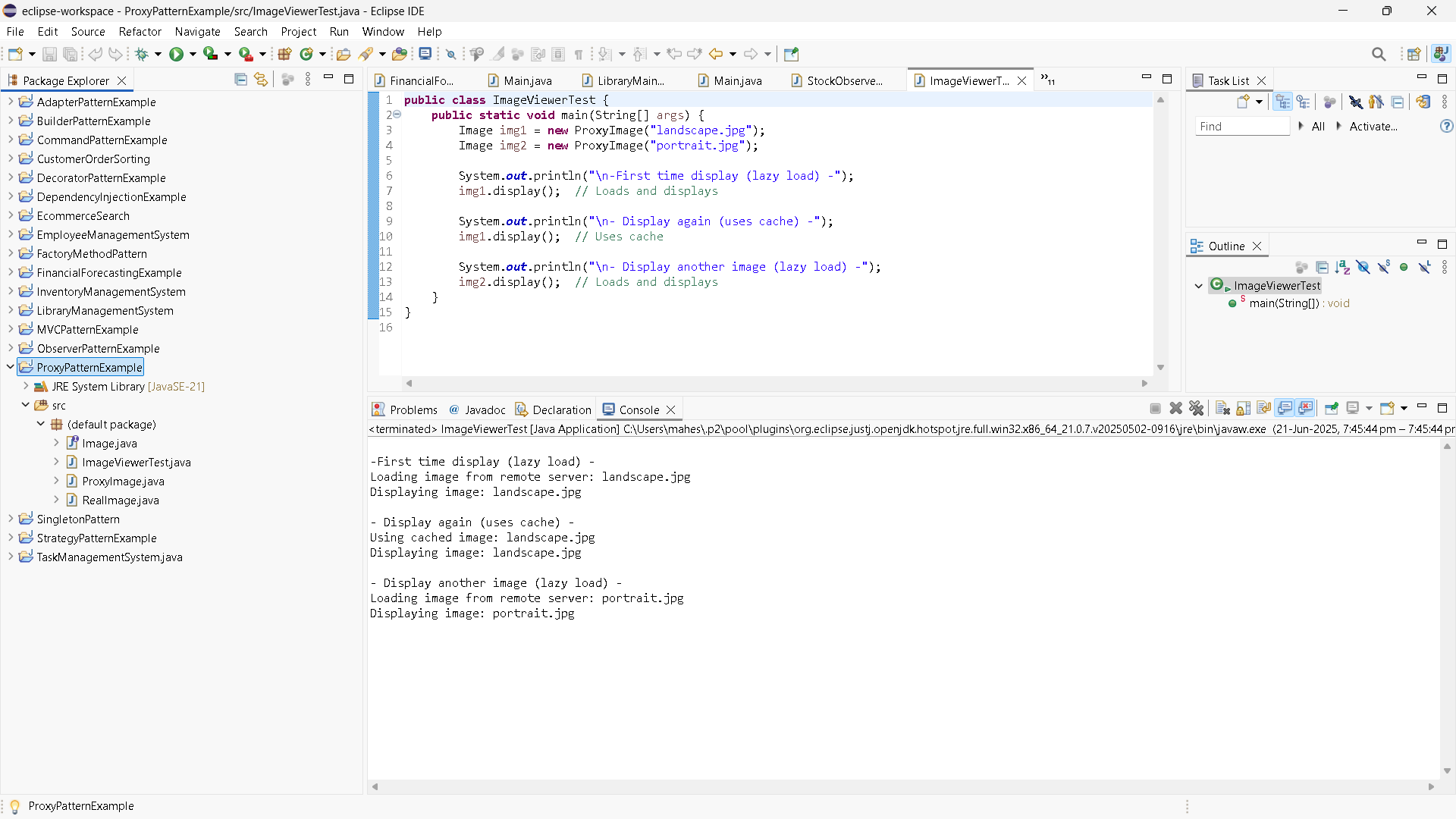
@Override

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

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

}

}



1. ObserverPatternExample

MobileApp.java

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

**private** String user;

**public** MobileApp(String user) {

**this**.user = user;

}

@Override

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

System.***out***.println(" [Mobile] " + user + " notified: " + stockSymbol + " is now Rs. " + newPrice);

}

}

Observer.java

**public** **interface** Observer {

**void** update(String stockSymbol, **double** newPrice);

}

Stock.java

**public** **interface** Stock {

**void** registerObserver(Observer o);

**void** removeObserver(Observer o);

**void** notifyObservers();

}

StockMarket.java

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** StockMarket **implements** Stock {

**private** List<Observer> observers;

**private** String stockSymbol;

**private** **double** stockPrice;

**public** StockMarket(String stockSymbol) {

**this**.stockSymbol = stockSymbol;

**this**.observers = **new** ArrayList<>();

}

**public** **void** setStockPrice(**double** newPrice) {

System.***out***.println("\n Price Update: " + stockSymbol + " -> Rs. " + newPrice);

**this**.stockPrice = newPrice;

notifyObservers();

}

@Override

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

observers.add(o);

}

@Override

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

observers.remove(o);

}

@Override

**public** **void** notifyObservers() {

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

o.update(stockSymbol, stockPrice);

}

}

}

StockMarket.java

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** StockMarket **implements** Stock {

**private** List<Observer> observers;

**private** String stockSymbol;

**private** **double** stockPrice;

**public** StockMarket(String stockSymbol) {

**this**.stockSymbol = stockSymbol;

**this**.observers = **new** ArrayList<>();

}

**public** **void** setStockPrice(**double** newPrice) {

System.***out***.println("\n Price Update: " + stockSymbol + " -> Rs. " + newPrice);

**this**.stockPrice = newPrice;

notifyObservers();

}

@Override

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

observers.add(o);

}

@Override

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

observers.remove(o);

}

@Override

**public** **void** notifyObservers() {

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

o.update(stockSymbol, stockPrice);

}

}

}

StockObserverTest.java

**public** **class** StockObserverTest {

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

StockMarket relianceStock = **new** StockMarket("RELIANCE");

Observer mobileUser = **new** MobileApp("Preeti");

Observer webUser = **new** WebApp("Vikram");

relianceStock.registerObserver(mobileUser);

relianceStock.registerObserver(webUser);

relianceStock.setStockPrice(2800.0);

relianceStock.setStockPrice(2850.5);

relianceStock.removeObserver(webUser);

relianceStock.setStockPrice(2900.0);

}

}

WebApp.java

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

**private** String user;

**public** WebApp(String user) {

**this**.user = user;

}

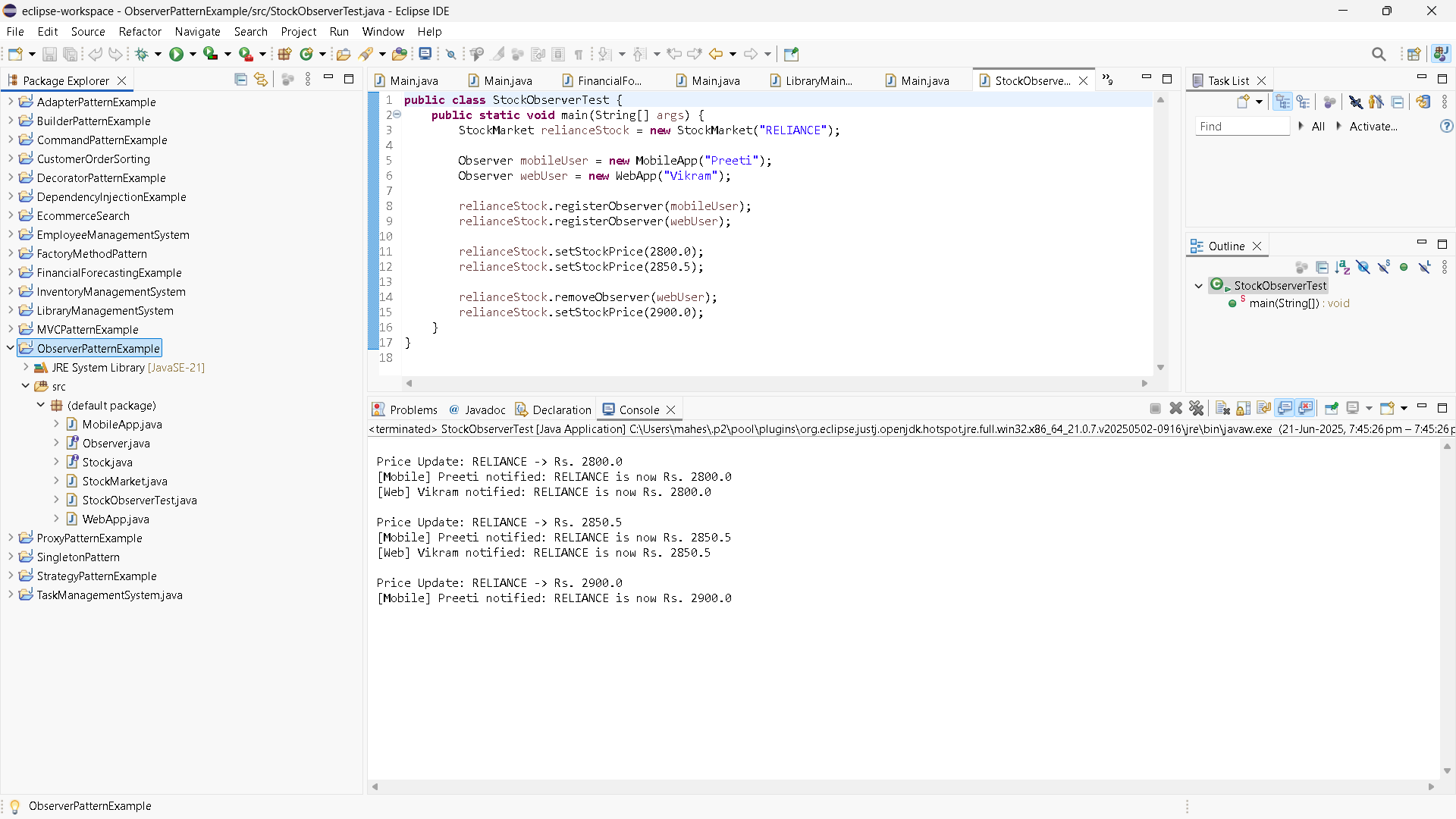
@Override

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

System.***out***.println(" [Web] " + user + " notified: " + stockSymbol + " is now Rs. " + newPrice);

}

}



1. Strategy Pattern

CreditCardPayment.java

//package strategy;

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

**private** String cardNumber;

**private** String cardHolder;

**public** CreditCardPayment(String cardNumber, String cardHolder) {

**this**.cardNumber = cardNumber;

**this**.cardHolder = cardHolder;

}

@Override

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

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

}

}

Main.java

**public** **class** Main {

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

PaymentContext context = **new** PaymentContext();

// Pay with Credit Card

context.setPaymentStrategy(**new** CreditCardPayment("1234-5678-9012-3456", "Yamini"));

context.payAmount(1500.00);

// Pay with PayPal

context.setPaymentStrategy(**new** PayPalPayment("Yamini@gmail.com"));

context.payAmount(750.00);

}

}

PaymentContext.java

//package strategy;

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

**private** PaymentStrategy paymentStrategy;

// Set strategy at runtime

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

**this**.paymentStrategy = paymentStrategy;

}

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

**if** (paymentStrategy == **null**) {

System.***out***.println("Payment strategy not set!");

} **else** {

paymentStrategy.pay(amount);

}

}

}

PaymentStrategy.java

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

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

}

PayPalPayment.java

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

**private** String email;

**public** PayPalPayment(String email) {

**this**.email = email;

}

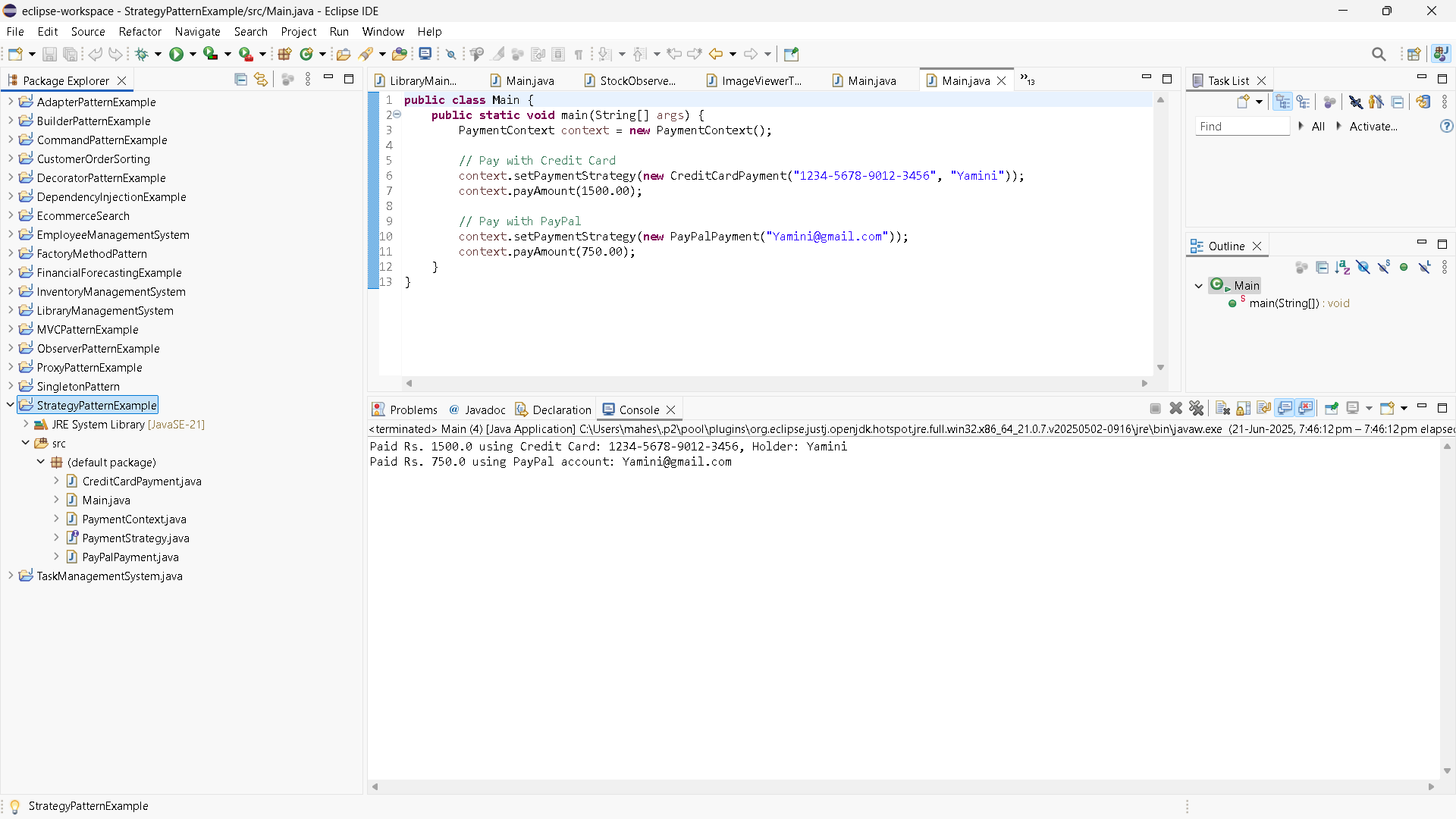
@Override

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

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

}

}



1. CommandPatternExample

Command.java

**package** commandpattern;

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

**void** execute();

}

Light.java

**package** commandpattern;

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

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

System.***out***.println("The light is ON");

}

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

System.***out***.println("The light is OFF");

}

}

LightOffCommand.java

**package** commandpattern;

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

**private** Light light;

**public** LightOffCommand(Light light) {

**this**.light = light;

}

@Override

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

light.turnOff();

}

}

LightOnCommand.java

**package** commandpattern;

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

**private** Light light;

**public** LightOnCommand(Light light) {

**this**.light = light;

}

@Override

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

light.turnOn();

}

}

Main.java

**package** commandpattern;

**public** **class** Main {

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

// Receiver

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

// Commands

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

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

// Invoker

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

// Turn light ON

remote.setCommand(lightOn);

remote.pressButton();

// Turn light OFF

remote.setCommand(lightOff);

remote.pressButton();

}

}

RemoteControl.java

**package** commandpattern;

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

**private** Command command;

**public** **void** setCommand(Command command) {

**this**.command = command;

}

**public** **void** pressButton() {

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

command.execute();

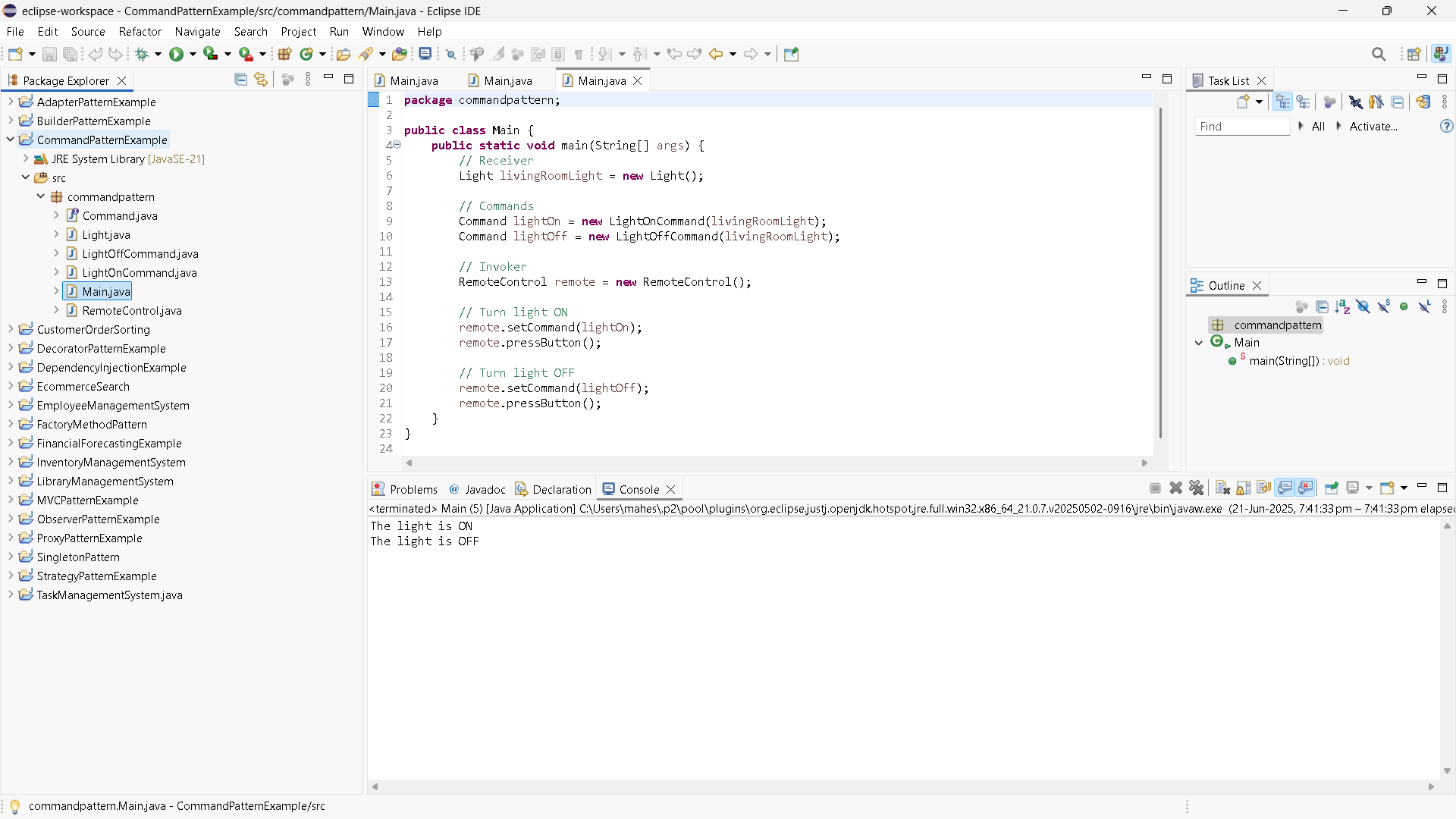
} **else** {

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

}

}

}



1. MVCPatternExample

Main.java

//package mvcpattern;

**public** **class** Main {

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

// Create the model

Student student = **new** Student("Shalini", "S101", "A");

// Create the view

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

// Create the controller

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

// Display initial details

controller.updateView();

// Update student details

controller.setStudentName("Shalini Pandey");

controller.setStudentGrade("A+");

// Display updated details

controller.updateView();

}

}

Student.java

//package mvcpattern;

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

**private** String name;

**private** String id;

**private** String grade;

// Constructor

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

**this**.name = name;

**this**.id = id;

**this**.grade = grade;

}

// Getters and Setters

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getId() {

**return** id;

}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** String getGrade() {

**return** grade;

}

**public** **void** setGrade(String grade) {

**this**.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;

}

// Controller methods to update model data

**public** **void** setStudentName(String name) {

model.setName(name);

}

**public** **void** setStudentId(String id) {

model.setId(id);

}

**public** **void** setStudentGrade(String grade) {

model.setGrade(grade);

}

// Controller methods to get model data

**public** String getStudentName() {

**return** model.getName();

}

**public** String getStudentId() {

**return** model.getId();

}

**public** String getStudentGrade() {

**return** model.getGrade();

}

// Update the view

**public** **void** updateView() {

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

}

}

StudentView.java

//package mvcpattern;

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

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

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

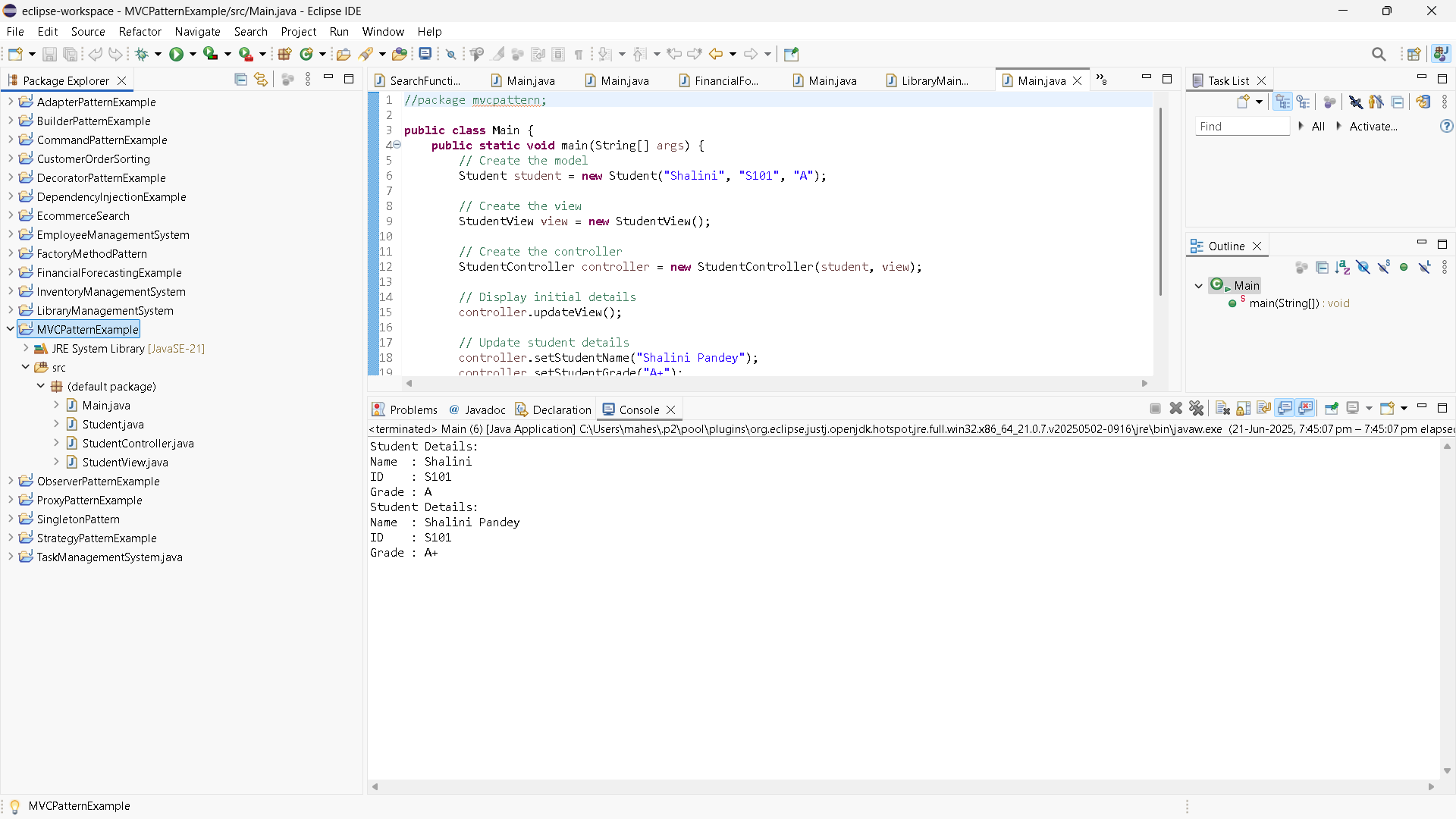
System.***out***.println("Name : " + name);

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

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

}

}



1. DependencyInjectionExample

Customer.java

//package di;

**public** **class** Customer {

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

**private** String name;

**private** String email;

**public** Customer(**int** id, String name, String email) {

**this**.id = id;

**this**.name = name;

**this**.email = email;

}

**public** String toString() {

**return** "Customer ID: " + id + ", Name: " + name + ", Email: " + email;

}

}

CustomerRepository.java

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

Customer findCustomerById(**int** id);

}

CustomerREpositoryImpl.java

//package di;

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

@Override

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

// Dummy data

**return** **new** Customer(id, "Preeti sharma", "preetisharma@gmail.com");

}

}

CustomerService.java

//package di;

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

**private** CustomerRepository customerRepository;

// Constructor injection

**public** CustomerService(CustomerRepository customerRepository) {

**this**.customerRepository = customerRepository;

}

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

Customer customer = customerRepository.findCustomerById(id);

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

}

}

Main.java

//package di;

**public** **class** Main {

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

// Dependency

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

// Inject dependency

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

// Use service

service.getCustomerDetails(101);

}

}

