**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).");**

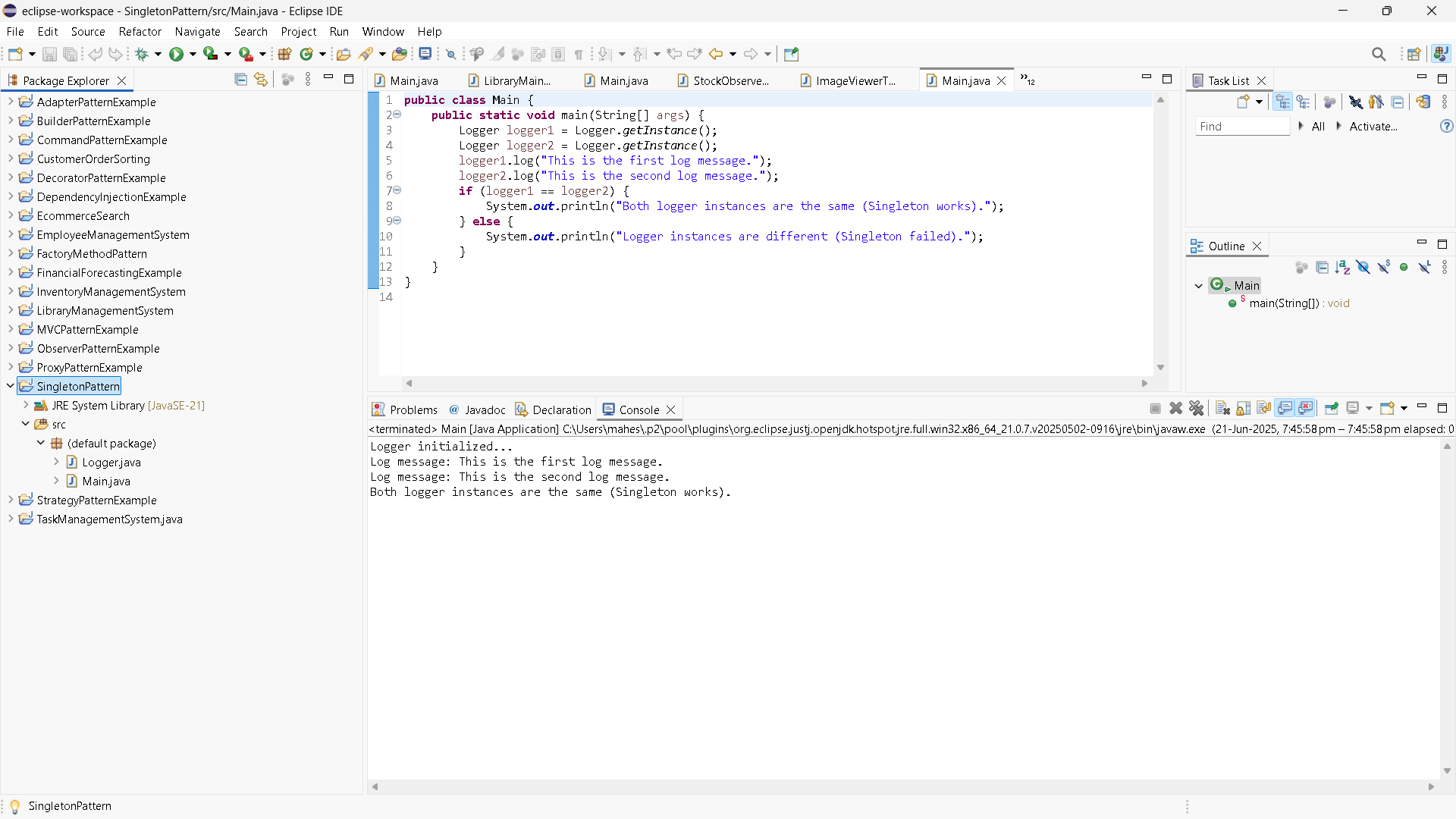
**} else {**

**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**

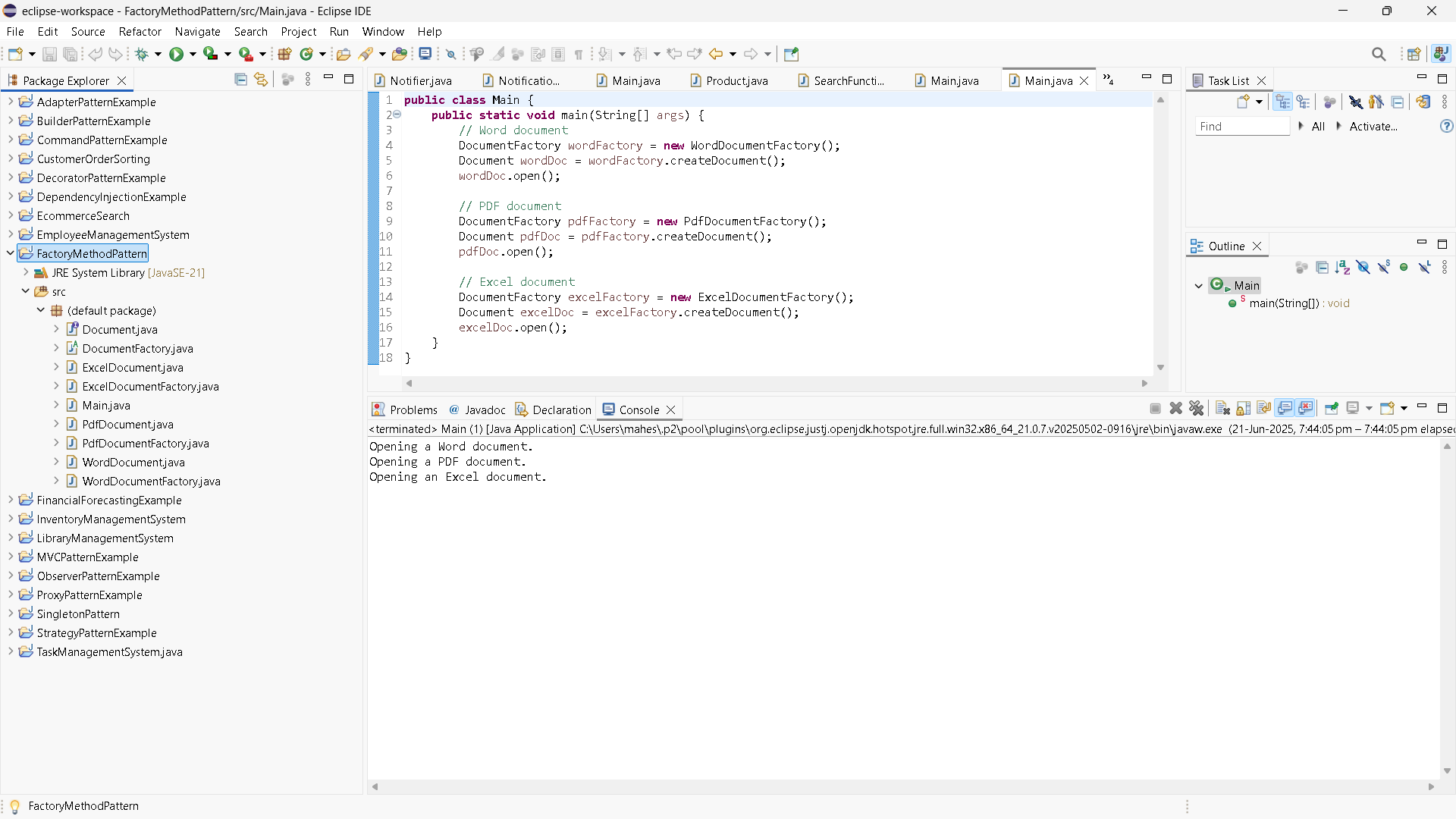
**DocumentFactory excelFactory = new ExcelDocumentFactory();**

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

**excelDoc.open();**

**}**

**}**



Data Structures and Algorithms

1. Ecommerce search

Product.java

**public class Product {**

**private int productId;**

**private String productName;**

**private String category;**

**public Product(int productId, String productName, String category) {**

**this.productId = productId;**

**this.productName = productName;**

**this.category = category;**

**}**

**public String getProductName() {**

**return productName;**

**}**

**@Override**

**public String toString() {**

**return "Product [ID=" + productId + ", Name=" + productName + ", Category=" + category + "]";**

**}**

**}**

SearchFunctions.java

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

**public class SearchFunctions {**

**// Linear search method**

**public static Product linearSearch(List<Product> products, String targetName) {**

**for (Product p : products) {**

**if (p.getProductName().equalsIgnoreCase(targetName)) {**

**return p;**

**}**

**}**

**return null;**

**}**

**// Binary search method**

**public static Product binarySearch(List<Product> products, String targetName) {**

**int low = 0;**

**int high = products.size() - 1;**

**while (low <= high) {**

**int mid = (low + high) / 2;**

**String midName = products.get(mid).getProductName().toLowerCase();**

**int cmp = midName.compareTo(targetName.toLowerCase());**

**if (cmp == 0)**

**return products.get(mid);**

**else if (cmp < 0)**

**low = mid + 1;**

**else**

**high = mid - 1;**

**}**

**return null;**

**}**

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

**List<Product> products = new ArrayList<>();**

**products.add(new Product(101, "Mobile", "Electronics"));**

**products.add(new Product(102, "Earpods", "Electronics"));**

**products.add(new Product(103, "Heels", "Apparel"));**

**products.add(new Product(104, "FashWash", "Beauty"));**

**// Sorting for binary search**

**products.sort(Comparator.*comparing*(p -> p.getProductName().toLowerCase()));**

**String searchQuery = "Earpods";**

**// Linear Search**

**Product linearResult = *linearSearch*(products, searchQuery);**

**System.*out*.println("Linear Search Result: " + (linearResult != null ? linearResult : "Product not found"));**

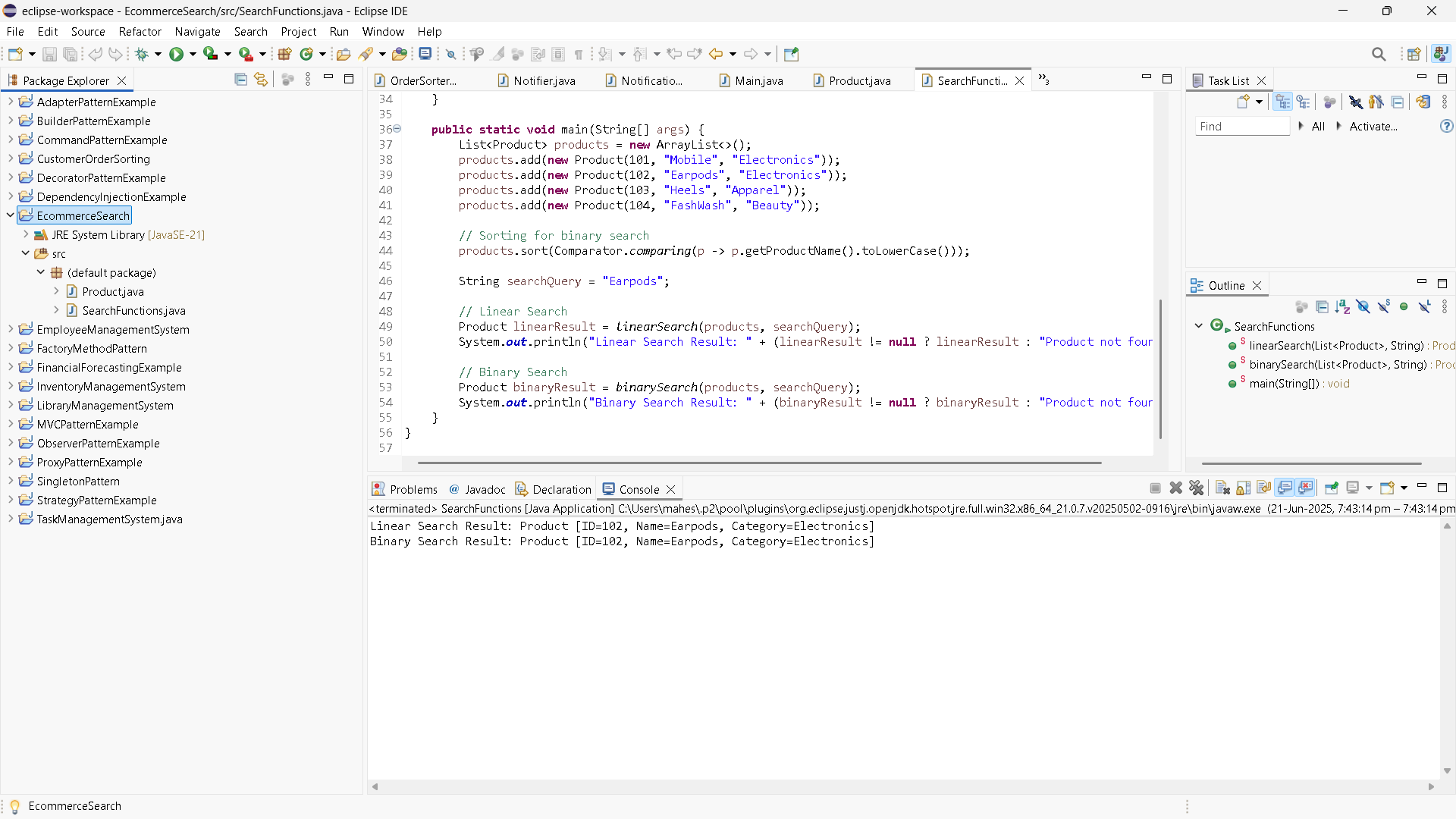
**// Binary Search**

**Product binaryResult = *binarySearch*(products, searchQuery);**

**System.*out*.println("Binary Search Result: " + (binaryResult != null ? binaryResult : "Product not found"));**

**}**

**}**



1. FinancialForecast.java

**public** **class** FinancialForecast {

**public** **static** **double** futureValue(**double** principal, **double** rate, **int** periods) {

**if** (periods == 0) {

**return** principal;

}

**return** *futureValue*(principal, rate, periods - 1) \* (1 + rate);

}

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

**double** presentValue = 10000.0; // Initial investment

**double** annualGrowthRate = 0.05; // 5% growth

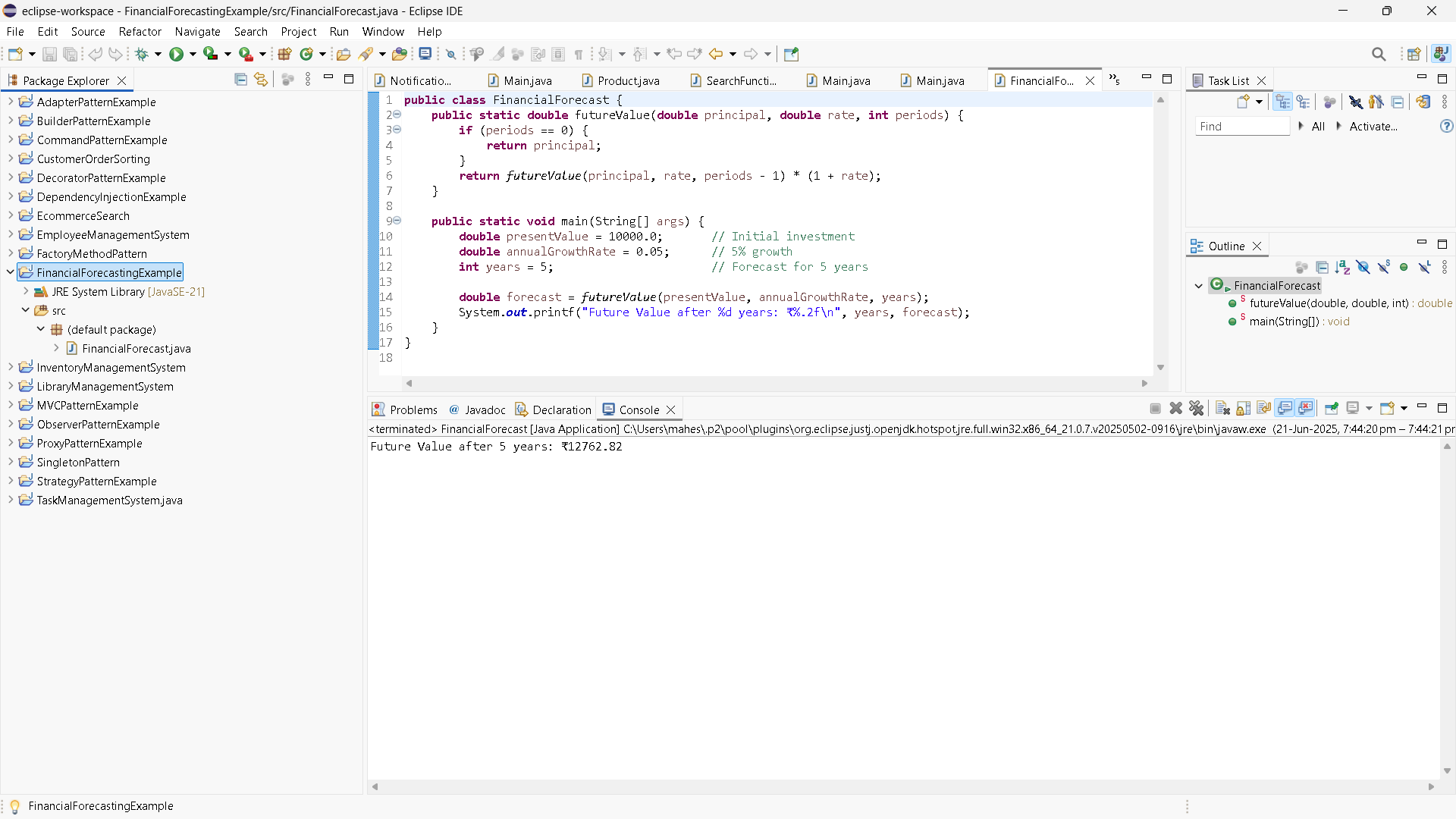
**int** years = 5; // Forecast for 5 years

**double** forecast = *futureValue*(presentValue, annualGrowthRate, years);

System.***out***.printf("Future Value after %d years: ₹%.2f\n", years, forecast);

}

}



Practice problems

Design Patterns And Principles

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")

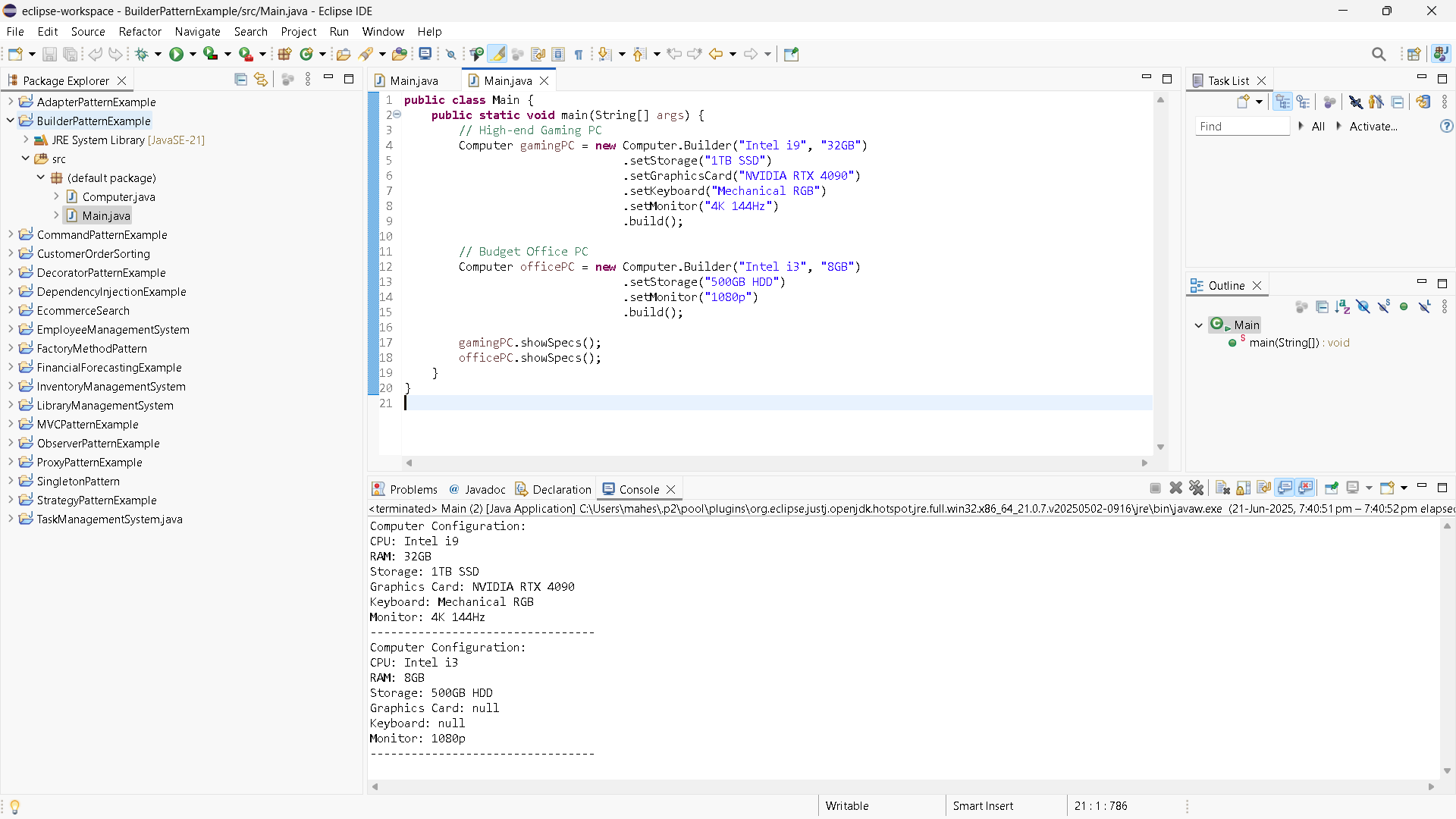
.build();

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

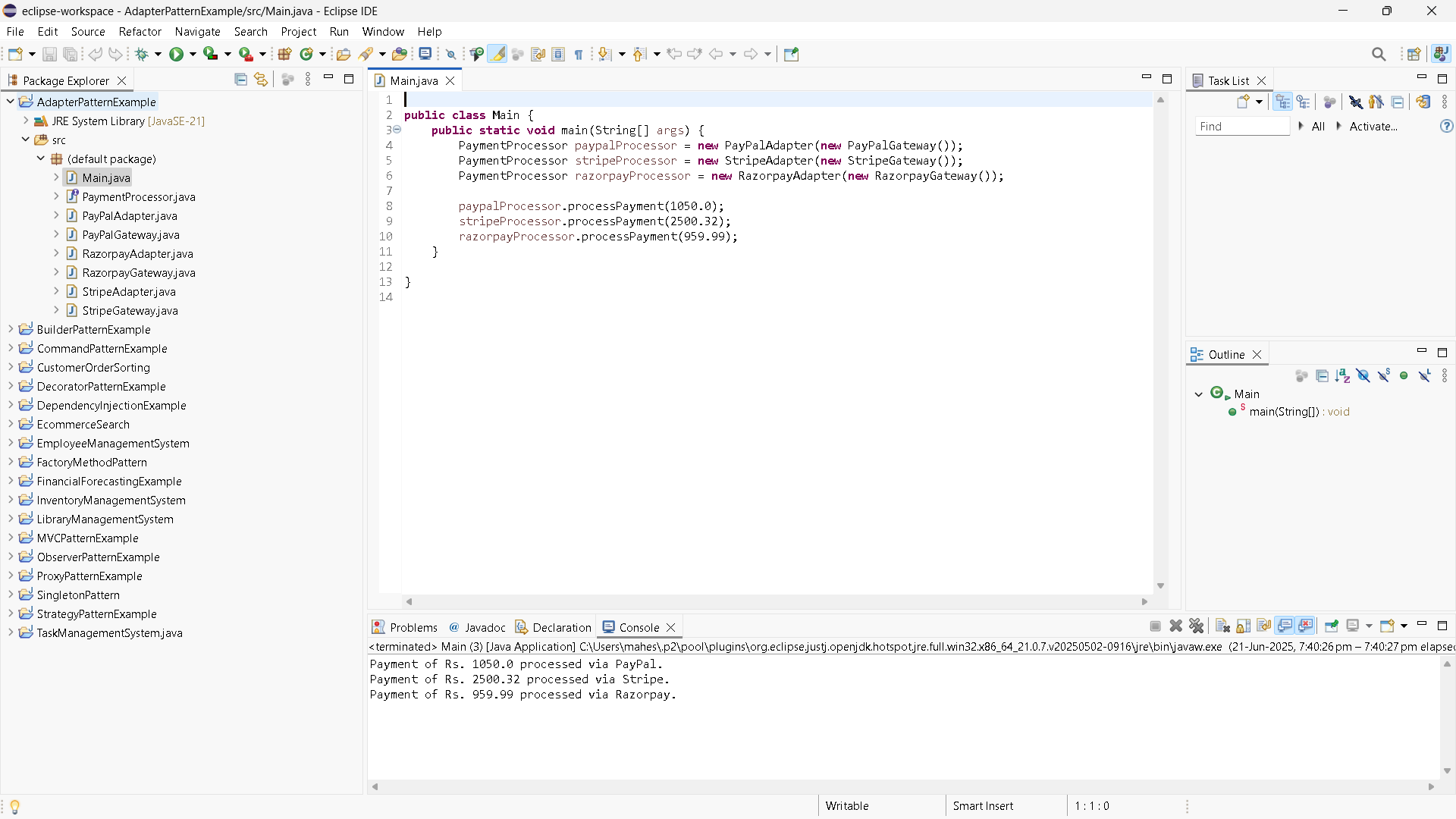
**public** **class** StripeGateway {

**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

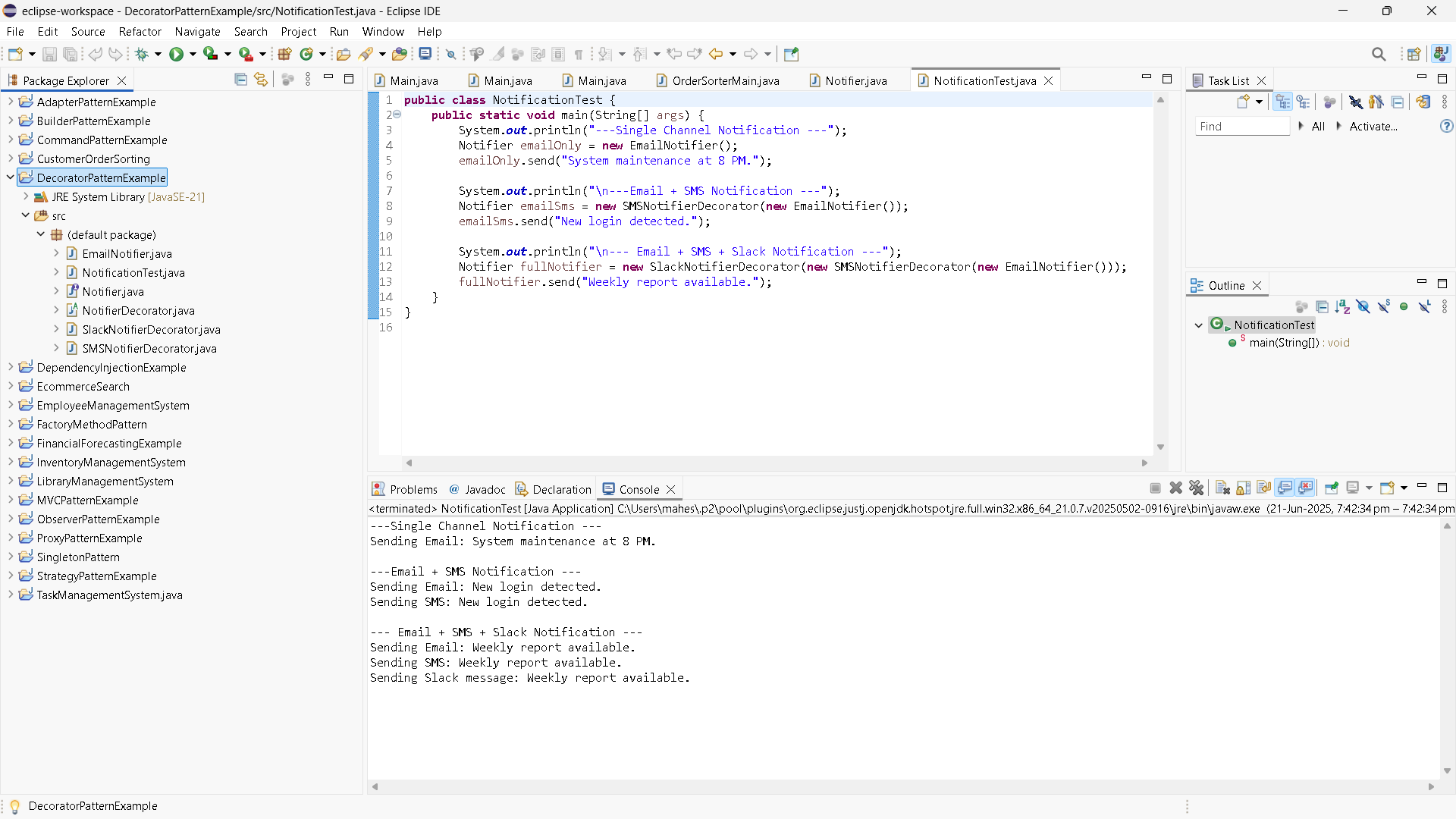
**public** **void** send(String message) {

**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);

}

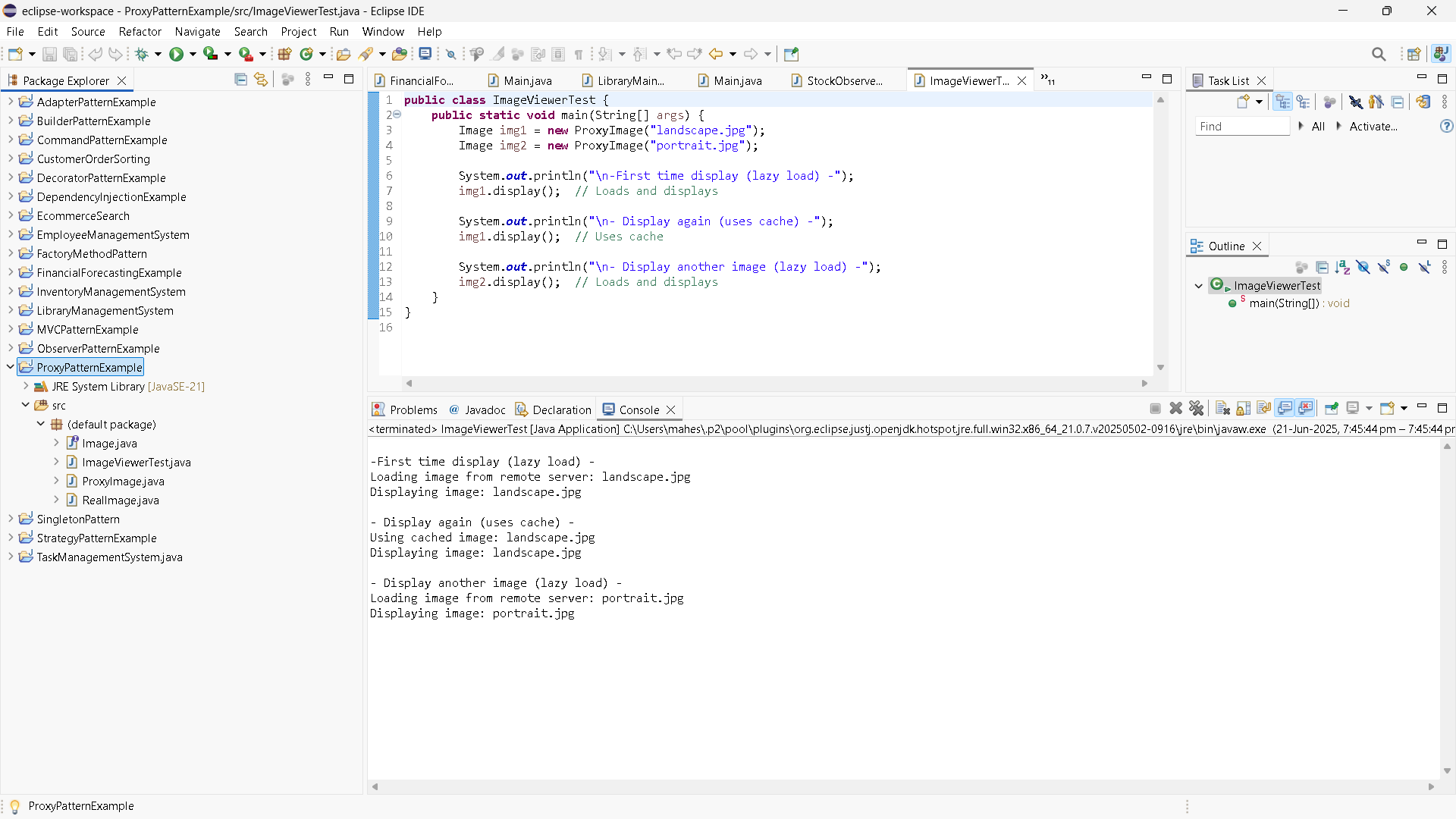
@Override

**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;

}

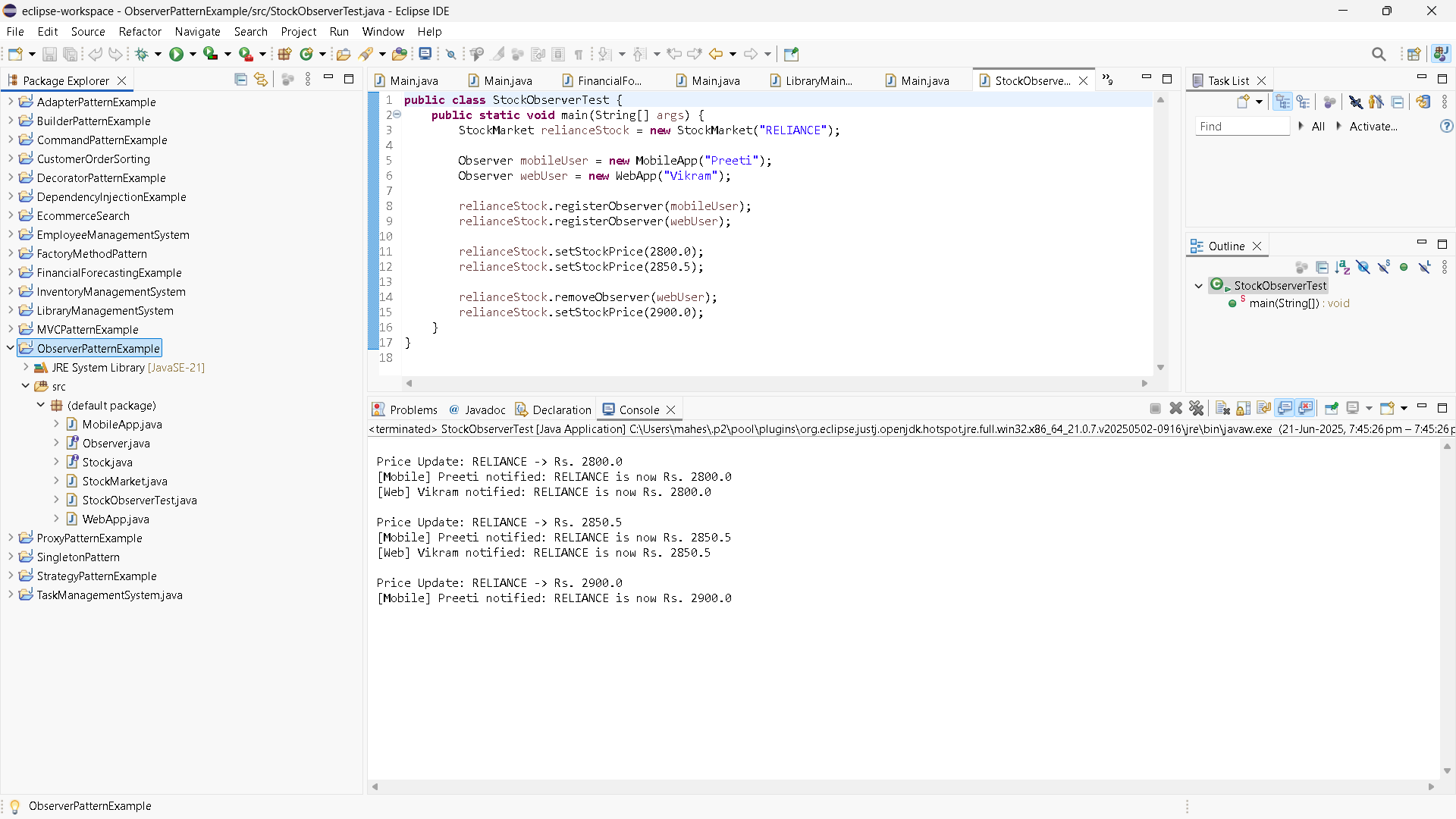
@Override

**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;

}

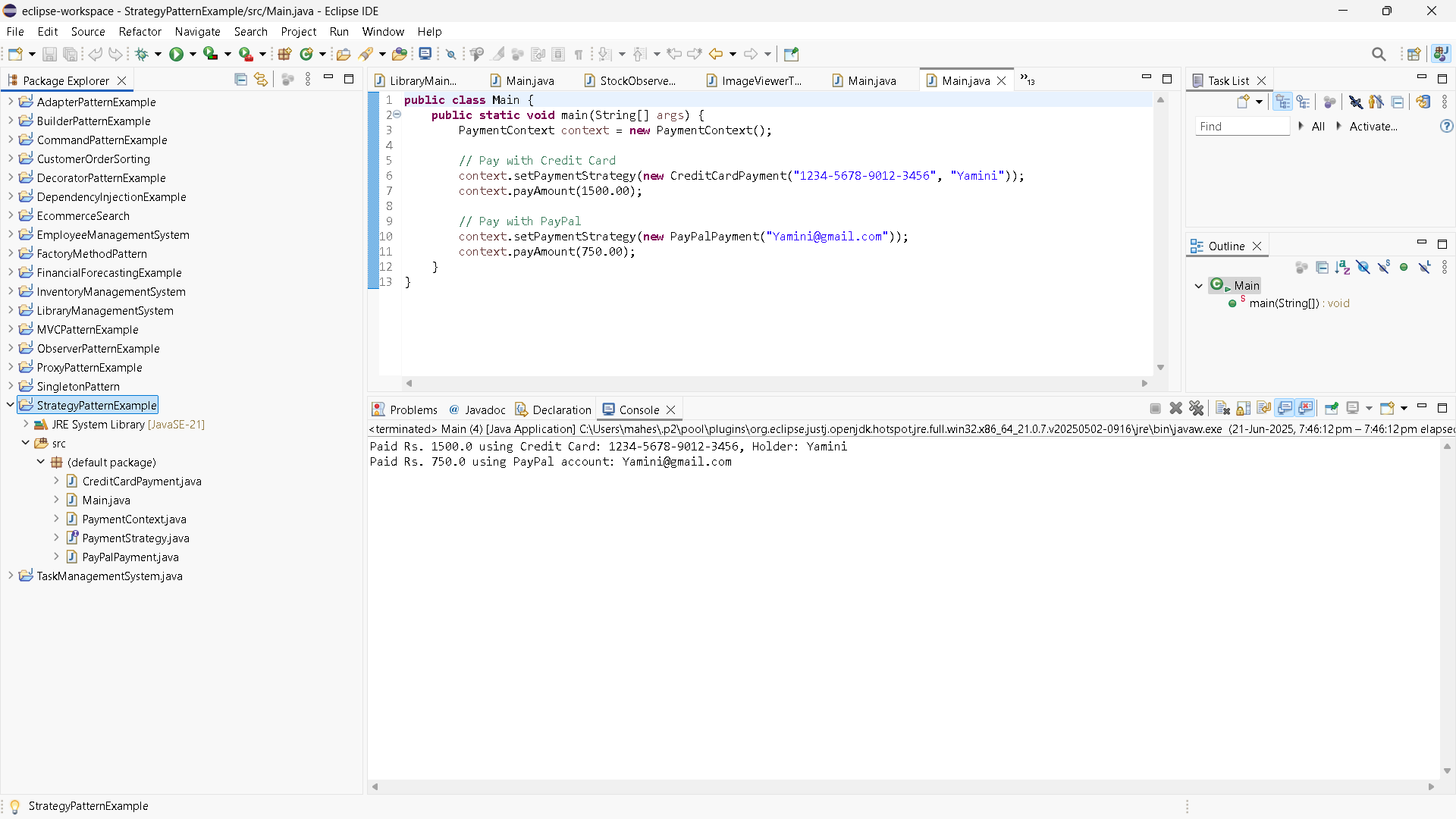
@Override

**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();

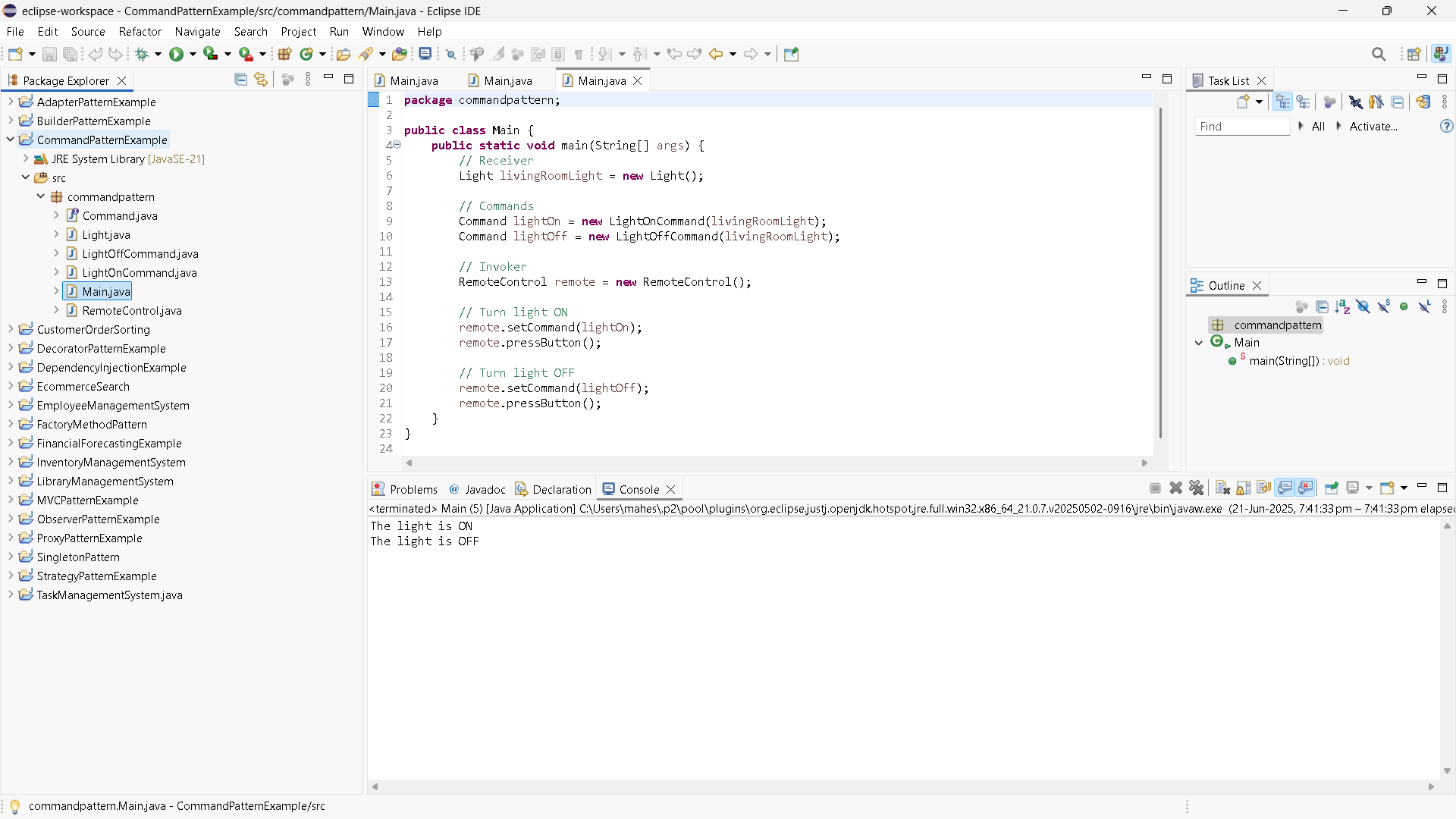
} **else** {

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:");

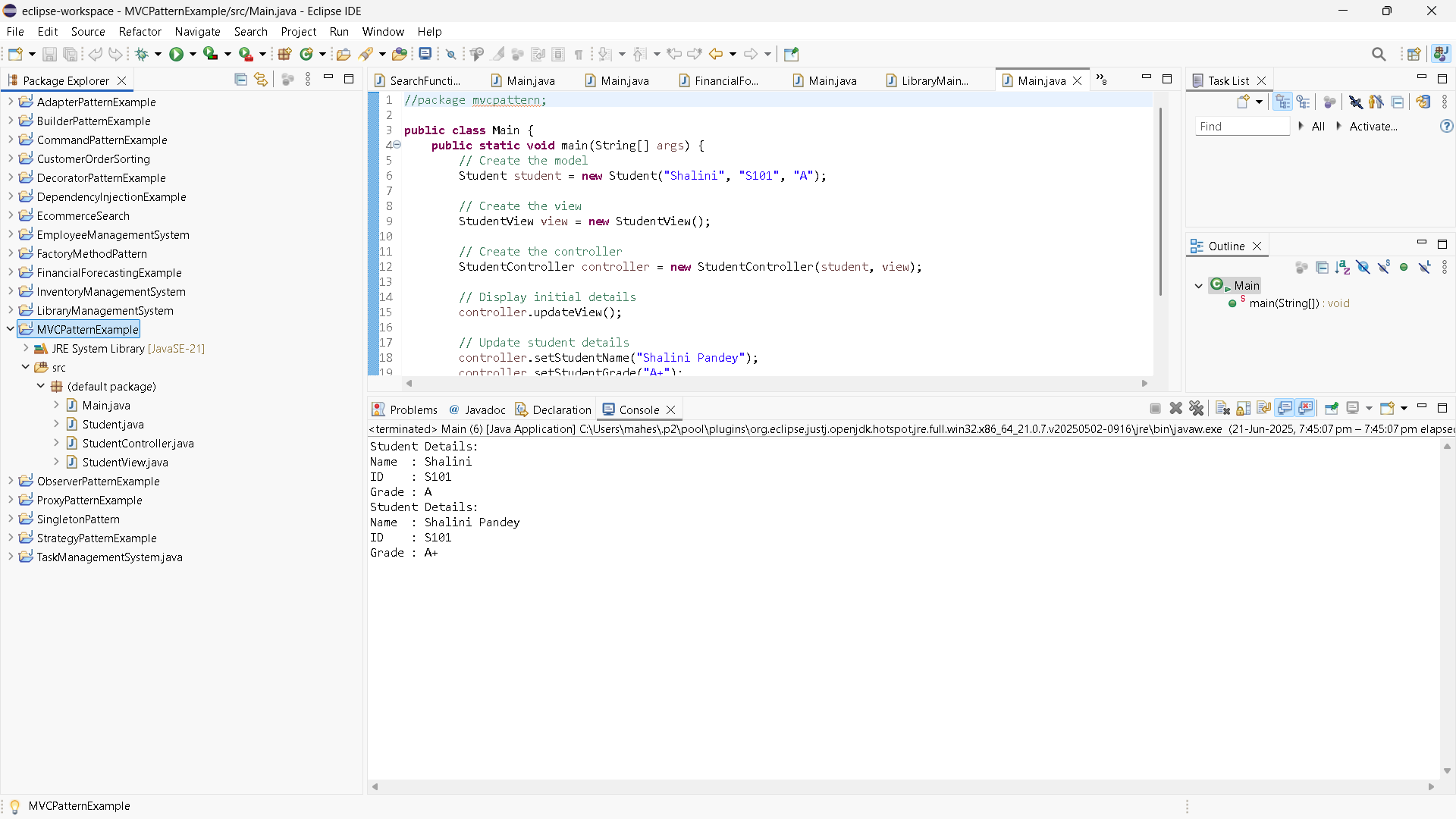
System.***out***.println("Name : " + name);

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

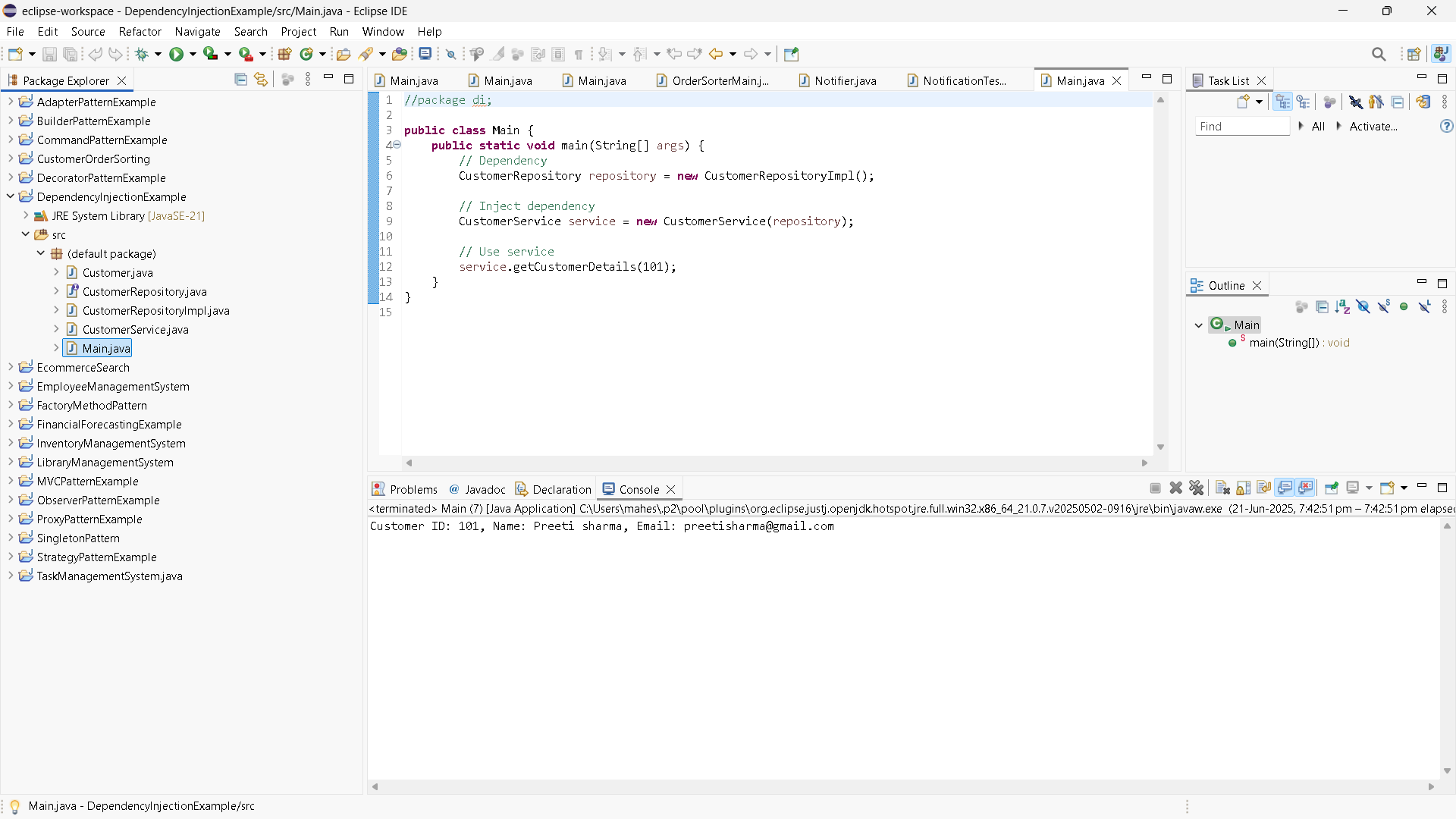
CustomerService service = **new** CustomerService(repository);

// Use service

service.getCustomerDetails(101);

}

}



Data structures and algorithms

1. InventoryManagementSystem

AddProductService.java

**package** inventory;

**public** **class** AddProductService {

**private** InventoryManager manager;

**public** AddProductService(InventoryManager manager) {

**this**.manager = manager;

}

**public** **void** execute(Product product) {

manager.addProduct(product);

}

}

DeleteProductService.java

**package** inventory;

**public** **class** DeleteProductService {

**private** InventoryManager manager;

**public** DeleteProductService(InventoryManager manager) {

**this**.manager = manager;

}

**public** **void** execute(**int** productId) {

manager.deleteProduct(productId);

}

}

InventoryManager.java

**package** inventory;

**import** java.util.HashMap;

**public** **class** InventoryManager {

**private** HashMap<Integer, Product> inventory = **new** HashMap<>();

**public** **void** addProduct(Product product) {

**if** (inventory.containsKey(product.getProductId())) {

System.***out***.println("Product ID already exists.");

} **else** {

inventory.put(product.getProductId(), product);

System.***out***.println("Product added.");

}

}

**public** **void** updateProduct(**int** productId, **int** newQty, **double** newPrice) {

Product product = inventory.get(productId);

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

product.setQuantity(newQty);

product.setPrice(newPrice);

System.***out***.println("Product updated.");

} **else** {

System.***out***.println("Product not found.");

}

}

**public** **void** deleteProduct(**int** productId) {

**if** (inventory.remove(productId) != **null**) {

System.***out***.println("Product removed.");

} **else** {

System.***out***.println("Product not found.");

}

}

**public** **void** displayInventory() {

**if** (inventory.isEmpty()) {

System.***out***.println("Inventory is empty.");

} **else** {

inventory.values().forEach(System.***out***::println);

}

}

}

Main.java

**package** inventory;

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

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

InventoryManager manager = **new** InventoryManager();

// Create service instances

AddProductService addService = **new** AddProductService(manager);

UpdateProductService updateService = **new** UpdateProductService(manager);

DeleteProductService deleteService = **new** DeleteProductService(manager);

// Add products

addService.execute(**new** Product(101, "Laptop", 10, 50000));

addService.execute(**new** Product(102, "Mouse", 50, 500));

addService.execute(**new** Product(103, "Keyboard", 30, 1500));

System.***out***.println("\n--- Initial Inventory ---");

manager.displayInventory();

// Update product

updateService.execute(102, 60, 550);

// Delete product

deleteService.execute(103);

System.***out***.println("\n--- Updated Inventory ---");

manager.displayInventory();

}

}

Product.java

**package** inventory;

**public** **class** Product {

**private** **int** productId;

**private** String productName;

**private** **int** quantity;

**private** **double** price;

**public** Product(**int** productId, String productName, **int** quantity, **double** price) {

**this**.productId = productId;

**this**.productName = productName;

**this**.quantity = quantity;

**this**.price = price;

}

**public** **int** getProductId() {

**return** productId;

}

**public** String getProductName() {

**return** productName;

}

**public** **int** getQuantity() {

**return** quantity;

}

**public** **double** getPrice() {

**return** price;

}

**public** **void** setQuantity(**int** quantity) {

**this**.quantity = quantity;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

@Override

**public** String toString() {

**return** productId + " | " + productName + " | Qty: " + quantity + " | Rs. " + price;

}

}

UpdateProductService.java

**package** inventory;

**public** **class** UpdateProductService {

**private** InventoryManager manager;

**public** UpdateProductService(InventoryManager manager) {

**this**.manager = manager;

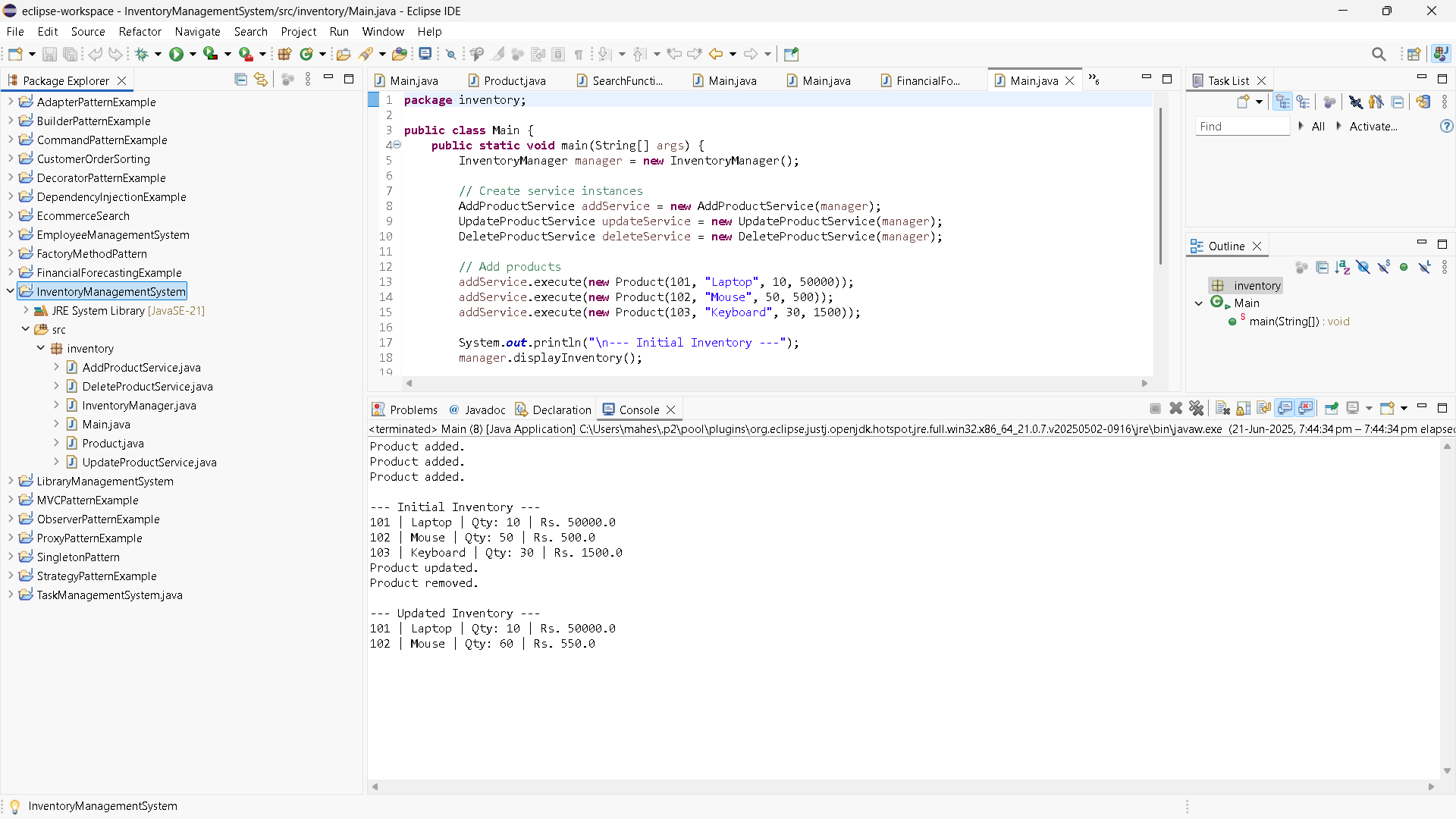
}

**public** **void** execute(**int** productId, **int** newQuantity, **double** newPrice) {

manager.updateProduct(productId, newQuantity, newPrice);

}

}



1. CustomerOrderSorting

BubbleSort.java

**public** **class** BubbleSort {

**public** **static** **void** sort(Order[] orders) {

**int** n = orders.length;

**for** (**int** i = 0; i < n - 1; i++) {

**for** (**int** j = 0; j < n - i - 1; j++) {

**if** (orders[j].totalPrice > orders[j + 1].totalPrice) {

Order temp = orders[j];

orders[j] = orders[j + 1];

orders[j + 1] = temp;

}

}

}

}

}

Order.java

**public** **class** Order {

**int** orderId;

String customerName;

**double** totalPrice;

**public** Order(**int** orderId, String customerName, **double** totalPrice) {

**this**.orderId = orderId;

**this**.customerName = customerName;

**this**.totalPrice = totalPrice;

}

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

System.***out***.println("Order ID: " + orderId + ", Customer: " + customerName + ", Total: RS. " + totalPrice);

}

}

OrderSorterMain.java

**public** **class** OrderSorterMain {

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

Order[] orders1 = {

**new** Order(101, "Yamini", 250.0),

**new** Order(102, "Priya", 100.0),

**new** Order(103, "Lasya", 400.0),

**new** Order(104, "Preeti", 150.0)

};

System.***out***.println("Original Orders:");

**for** (Order o : orders1) o.display();

System.***out***.println("\nSorted by Bubble Sort:");

BubbleSort.*sort*(orders1);

**for** (Order o : orders1) o.display();

Order[] orders2 = {

**new** Order(101, "Yamini", 250.0),

**new** Order(102, "Priya", 100.0),

**new** Order(103, "Lasya", 400.0),

**new** Order(104, "Preeti", 150.0)

};

System.***out***.println("\nSorted by Quick Sort:");

QuickSort.*sort*(orders2, 0, orders2.length - 1);

**for** (Order o : orders2) o.display();

}

}

QuickSort.java

**public** **class** QuickSort {

**public** **static** **void** sort(Order[] orders, **int** low, **int** high) {

**if** (low < high) {

**int** pi = *partition*(orders, low, high);

*sort*(orders, low, pi - 1);

*sort*(orders, pi + 1, high);

}

}

**private** **static** **int** partition(Order[] orders, **int** low, **int** high) {

**double** pivot = orders[high].totalPrice;

**int** i = low - 1;

**for** (**int** j = low; j < high; j++) {

**if** (orders[j].totalPrice <= pivot) {

i++;

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

Order temp = orders[i + 1];

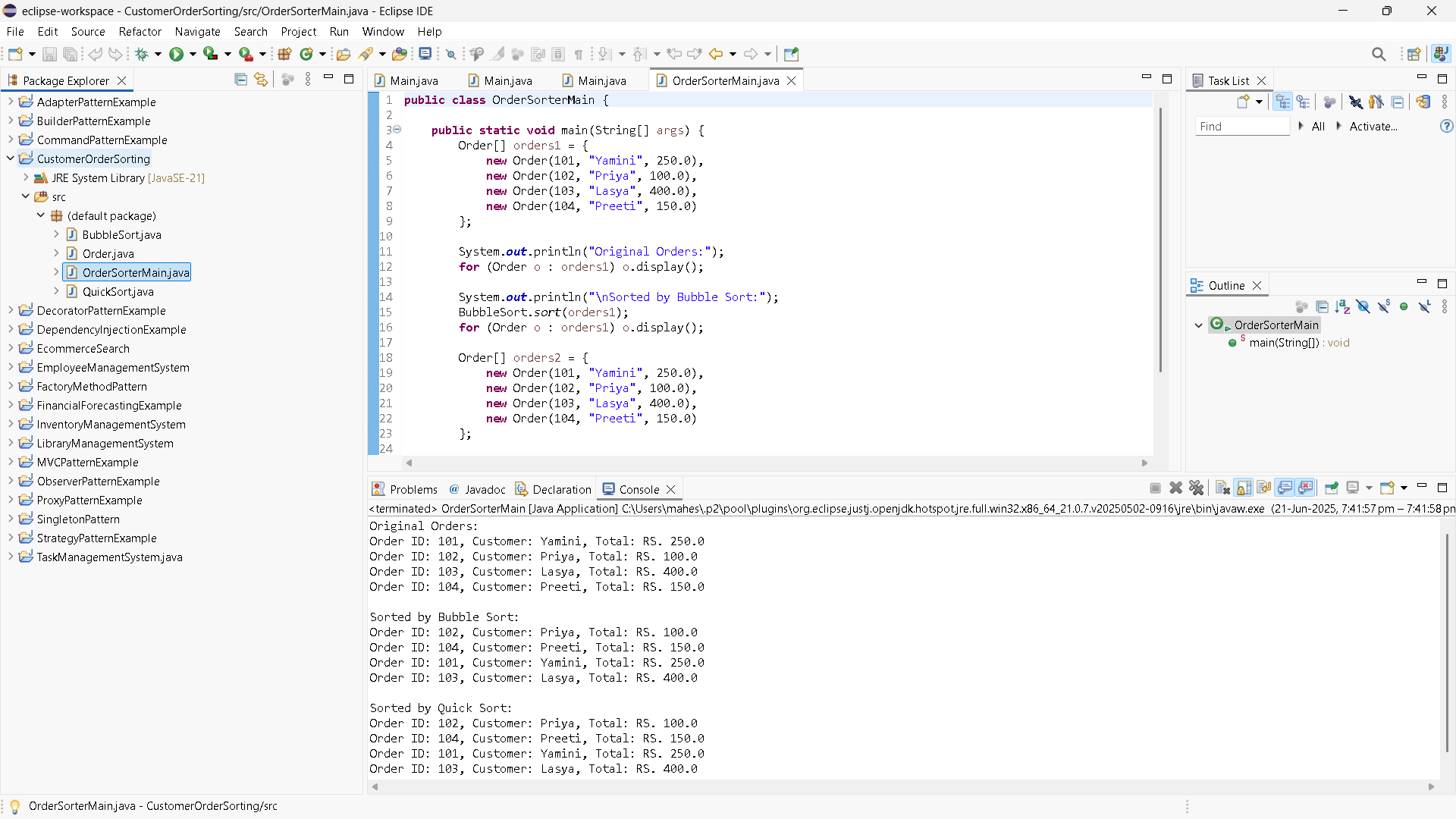
orders[i + 1] = orders[high];

orders[high] = temp;

**return** i + 1;

}

}



1. EmployeeManagementSystem

AddEmployee.java

**public** **class** AddEmployee {

**public** **static** **void** add(Employee[] employees, **int**[] count, Employee emp) {

**if** (count[0] < employees.length) {

employees[count[0]++] = emp;

System.***out***.println("Employee added.");

} **else** {

System.***out***.println("Employee list is full!");

}

}

}

DeleteEmployee.java

**public** **class** DeleteEmployee {

**public** **static** **void** delete(Employee[] employees, **int**[] count, **int** id) {

**for** (**int** i = 0; i < count[0]; i++) {

**if** (employees[i].employeeId == id) {

**for** (**int** j = i; j < count[0] - 1; j++) {

employees[j] = employees[j + 1];

}

employees[--count[0]] = **null**;

System.***out***.println("Employee deleted.");

**return**;

}

}

System.***out***.println("Employee not found.");

}

}

Employee.java

**public** **class** Employee {

**int** employeeId;

String name;

String position;

**double** salary;

**public** Employee(**int** employeeId, String name, String position, **double** salary) {

**this**.employeeId = employeeId;

**this**.name = name;

**this**.position = position;

**this**.salary = salary;

}

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

System.***out***.println("ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: $" + salary);

}

}

Main.java

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

**static** **final** **int** ***MAX*** = 100;

**static** Employee[] *employees* = **new** Employee[***MAX***];

**static** **int**[] *count* = {0}; // use array for pass-by-reference behavior

**public** **static** **void** displayAll() {

**if** (*count*[0] == 0) {

System.***out***.println("No employees to display.");

**return**;

}

**for** (**int** i = 0; i < *count*[0]; i++) {

*employees*[i].display();

}

}

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

AddEmployee.*add*(*employees*, *count*, **new** Employee(101, "Shalini", "Manager", 70000));

AddEmployee.*add*(*employees*, *count*, **new** Employee(102, "Rishitha", "Engineer", 50000));

AddEmployee.*add*(*employees*, *count*, **new** Employee(103, "Kumar", "HR", 45000));

System.***out***.println("\nAll Employees:");

*displayAll*();

System.***out***.println("\nSearching for ID 102:");

Employee found = SearchEmployee.*search*(*employees*, *count*[0], 102);

**if** (found != **null**) found.display();

**else** System.***out***.println("Employee not found.");

System.***out***.println("\nUpdating ID 103:");

UpdateEmployee.*update*(*employees*, *count*[0], 103, "Charles", "HR Lead", 48000);

System.***out***.println("\nDeleting ID 101:");

DeleteEmployee.*delete*(*employees*, *count*, 101);

System.***out***.println("\nAll Employees After Update & Delete:");

*displayAll*();

}

}

SearchEmployee.java

**public** **class** SearchEmployee {

**public** **static** Employee search(Employee[] employees, **int** count, **int** id) {

**for** (**int** i = 0; i < count; i++) {

**if** (employees[i].employeeId == id) {

**return** employees[i];

}

}

**return** **null**;

}

}

UpdateEmployee.java

**public** **class** UpdateEmployee {

**public** **static** **void** update(Employee[] employees, **int** count, **int** id, String newName, String newPosition, **double** newSalary) {

**for** (**int** i = 0; i < count; i++) {

**if** (employees[i].employeeId == id) {

employees[i].name = newName;

employees[i].position = newPosition;

employees[i].salary = newSalary;

System.***out***.println("Employee updated.");

**return**;

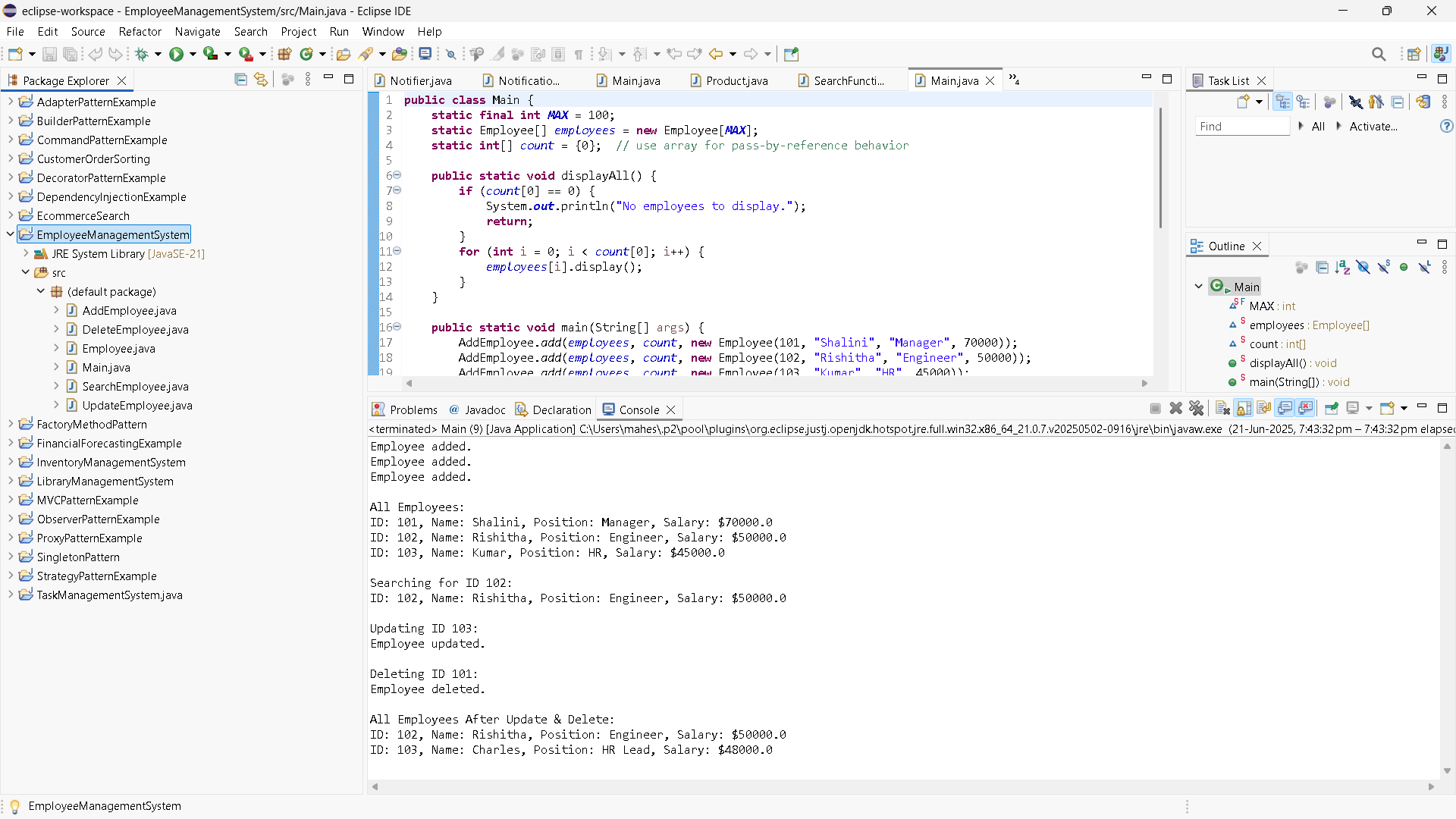
}

}

System.***out***.println("Employee not found.");

}

}



1. TaskManagementSystem

AddTask.java

**public** **class** AddTask {

**public** **static** **void** add(TaskList list, **int** taskId, String taskName, String status) {

Task newTask = **new** Task(taskId, taskName, status);

**if** (list.head == **null**) {

list.head = newTask;

} **else** {

Task current = list.head;

**while** (current.next != **null**) {

current = current.next;

}

current.next = newTask;

}

System.***out***.println("Task added: " + taskName);

}

}

DeleteTask.java

**public** **class** DeleteTask {

**public** **static** **void** delete(TaskList list, **int** id) {

**if** (list.head == **null**) {

System.***out***.println("Task list is empty.");

**return**;

}

**if** (list.head.taskId == id) {

list.head = list.head.next;

System.***out***.println("Task deleted.");

**return**;

}

Task current = list.head;

**while** (current.next != **null** && current.next.taskId != id) {

current = current.next;

}

**if** (current.next == **null**) {

System.***out***.println("Task not found.");

} **else** {

current.next = current.next.next;

System.***out***.println("Task deleted.");

}

}

}

DisplayTask.java

**public** **class** DisplayTasks {

**public** **static** **void** display(TaskList list) {

**if** (list.head == **null**) {

System.***out***.println("No tasks available.");

**return**;

}

Task current = list.head;

**while** (current != **null**) {

current.display();

current = current.next;

}

}

}

SearchTassk.java

**public** **class** SearchTask {

**public** **static** Task search(TaskList list, **int** id) {

Task current = list.head;

**while** (current != **null**) {

**if** (current.taskId == id) {

**return** current;

}

current = current.next;

}

**return** **null**;

}

}

Task.java

**public** **class** Task {

**int** taskId;

String taskName;

String status;

Task next;

**public** Task(**int** taskId, String taskName, String status) {

**this**.taskId = taskId;

**this**.taskName = taskName;

**this**.status = status;

**this**.next = **null**;

}

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

System.***out***.println("Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status);

}

}

TaskList.java

**public** **class** TaskList {

**public** Task head = **null**;

}

TaskManagerMain.java

**public** **class** TaskManagerMain {

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

TaskList list = **new** TaskList();

AddTask.*add*(list, 1, "Design Database", "Pending");

AddTask.*add*(list, 2, "Develop API", "In Progress");

AddTask.*add*(list, 3, "UI Integration", "Not Started");

System.***out***.println("\nAll Tasks:");

DisplayTasks.*display*(list);

System.***out***.println("\nSearching for Task ID 2:");

Task found = SearchTask.*search*(list, 2);

**if** (found != **null**) found.display();

**else** System.***out***.println("Task not found.");

System.***out***.println("\nDeleting Task ID 1:");

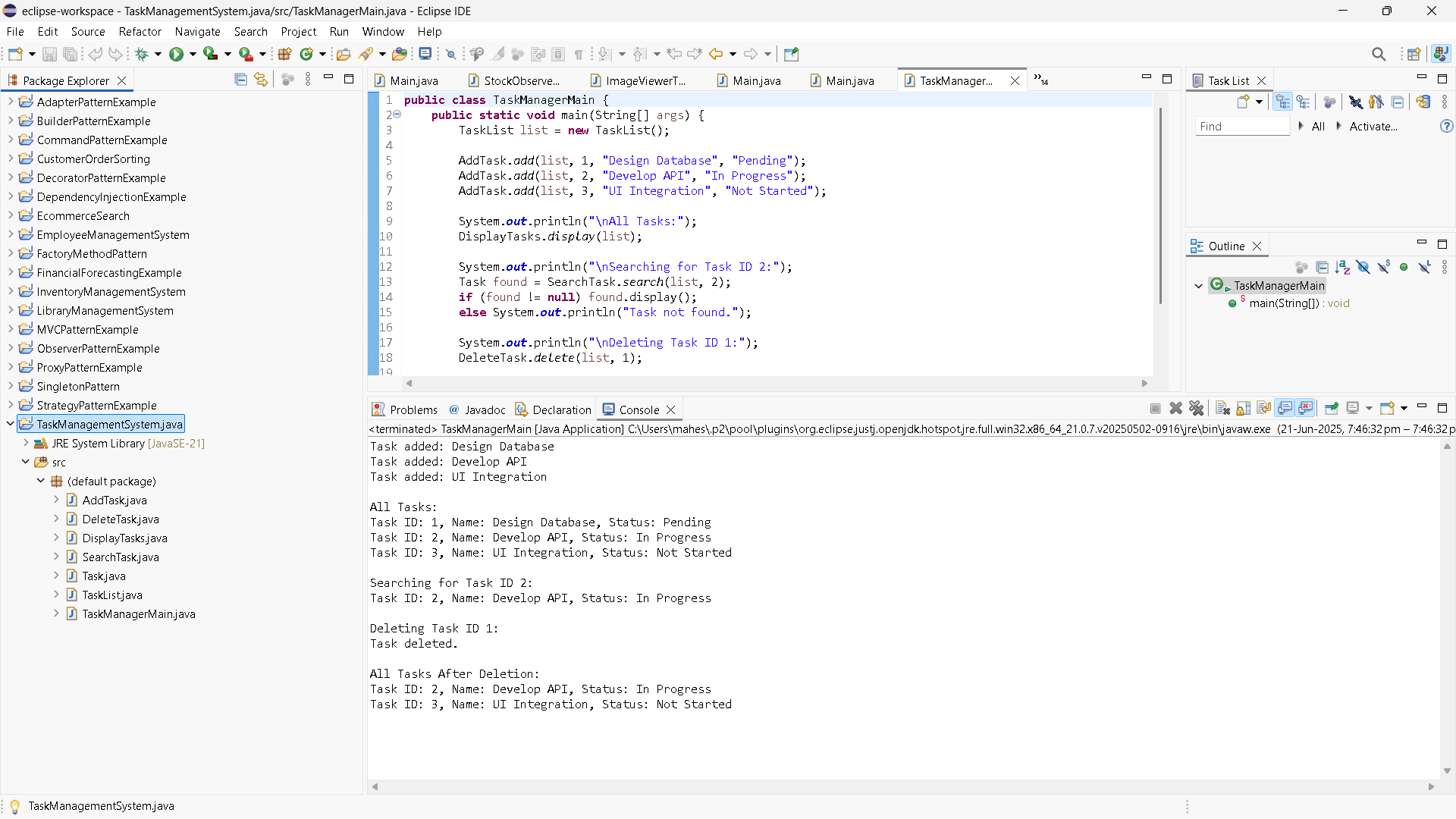
DeleteTask.*delete*(list, 1);

System.***out***.println("\nAll Tasks After Deletion:");

DisplayTasks.*display*(list);

}

}



6. LibraryManagementSystem

BinarySearch.java

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** BinarySearch {

// Sort books by title

**public** **static** **void** sortBooks(Book[] books) {

Arrays.*sort*(books, Comparator.*comparing*(b -> b.title.toLowerCase()));

}

// Binary search by title

**public** **static** Book search(Book[] books, String title) {

**int** left = 0, right = books.length - 1;

**while** (left <= right) {

**int** mid = (left + right) / 2;

**int** cmp = books[mid].title.compareToIgnoreCase(title);

**if** (cmp == 0) **return** books[mid];

**else** **if** (cmp < 0) left = mid + 1;

**else** right = mid - 1;

}

**return** **null**;

}

}

Book.java

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** BinarySearch {

// Sort books by title

**public** **static** **void** sortBooks(Book[] books) {

Arrays.*sort*(books, Comparator.*comparing*(b -> b.title.toLowerCase()));

}

// Binary search by title

**public** **static** Book search(Book[] books, String title) {

**int** left = 0, right = books.length - 1;

**while** (left <= right) {

**int** mid = (left + right) / 2;

**int** cmp = books[mid].title.compareToIgnoreCase(title);

**if** (cmp == 0) **return** books[mid];

**else** **if** (cmp < 0) left = mid + 1;

**else** right = mid - 1;

}

**return** **null**;

}

}

LibraryMain.java

**public** **class** LibraryMain {

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

Book[] books = {

**new** Book(101, "The Alchemist", "Paulo Coelho"),

**new** Book(102, "Atomic Habits", "James Clear"),

**new** Book(103, "Clean Code", "Robert C. Martin"),

**new** Book(104, "Deep Work", "Cal Newport"),

**new** Book(105, "Thinking Fast and Slow", "Daniel Kahneman")

};

System.***out***.println("Linear Search for 'Clean Code':");

Book found1 = LinearSearch.*search*(books, "Clean Code");

**if** (found1 != **null**) found1.display();

**else** System.***out***.println("Book not found.");

System.***out***.println("\n Sorting books for Binary Search...");

BinarySearch.*sortBooks*(books);

System.***out***.println(" Binary Search for 'Atomic Habits':");

Book found2 = BinarySearch.*search*(books, "Atomic Habits");

**if** (found2 != **null**) found2.display();

**else** System.***out***.println("Book not found.");

}

}

LinearSearch.java

**public** **class** LinearSearch {

**public** **static** Book search(Book[] books, String title) {

**for** (Book book : books) {

**if** (book.title.equalsIgnoreCase(title)) {

**return** book;

}

}

**return** **null**;

}

}

