DESIGN PATTERNS AND PRINCIPLES:

1: **Implementing the Singleton Pattern; Hands On.**

**CODE:**

**singletonPatternExample:**

Logger.java:

**package** singletonPatternExample;

**public** **class** Logger {

**private** **static** Logger *instance*;

**private** Logger() {

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

}

**public** **static** Logger getInstance() {

**if** (*instance* == **null**) {

*instance* = **new** Logger();

}

**return** *instance*;

}

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

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

}

}

Main.java:

**package** singletonPatternExample;

**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** {

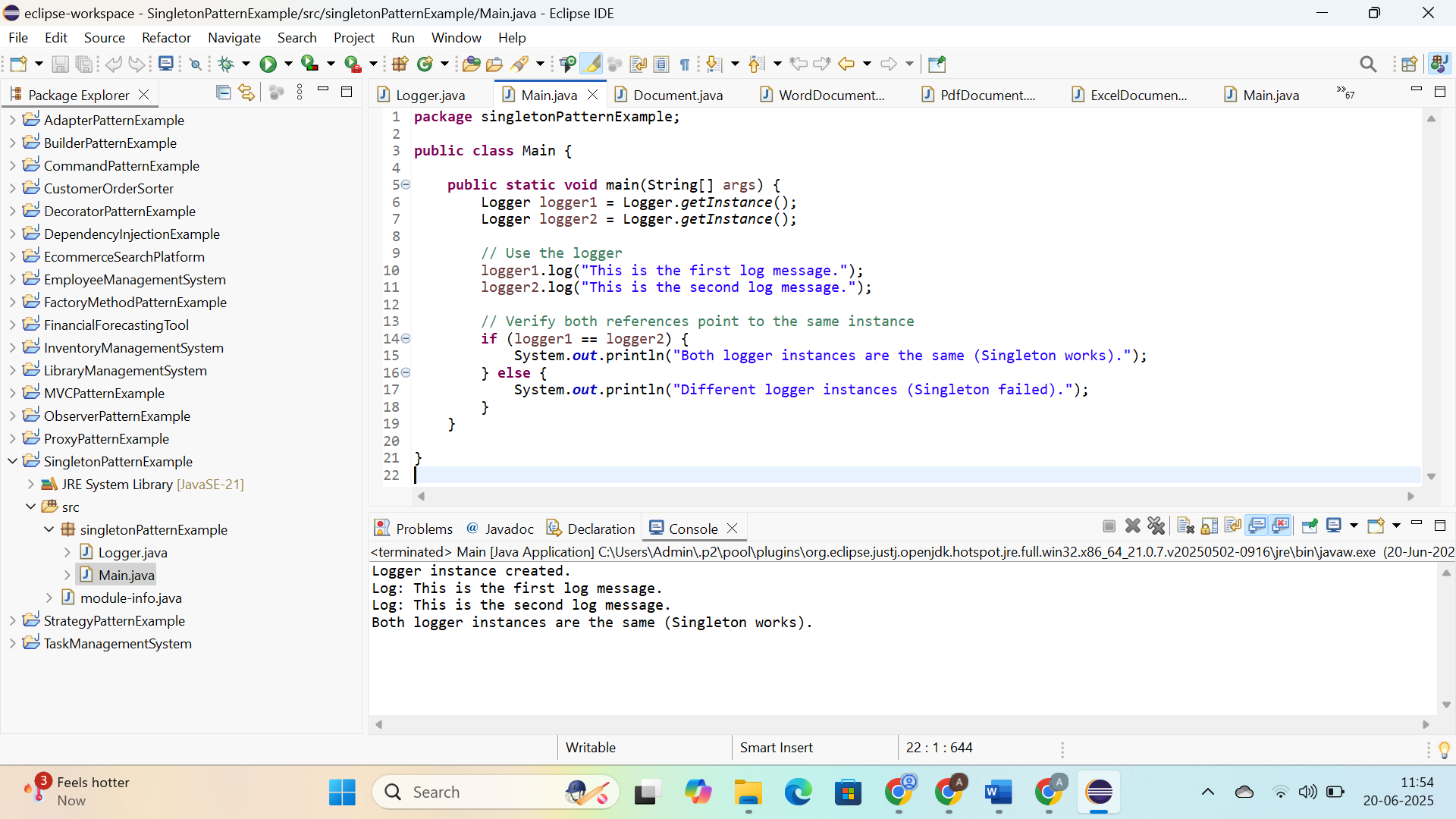
System.***out***.println("Different logger instances (Singleton failed).");

}

}

}

OUTPUT:



2: **Implementing the Factory Method Pattern; Hands On:**

**CODE:**

**document:**

Document.java:

**package** document;

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

**void** open();

}

ExcelDocument.java:

**package** document;

**public** **class** ExcelDocument **implements** Document {

**public** **void** open() {

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

}

}

PdfDocument.java:

**package** document;

**public** **class** PdfDocument **implements** Document {

**public** **void** open() {

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

}

}

WordDocument.java:

**package** document;

**public** **class** WordDocument **implements** Document {

**public** **void** open() {

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

}

}

factory:

DocumentFactory.java:

**package** factory;

**import** document.Document;

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

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

}

ExcelDocumentFactory.java:

**package** factory;

**import** document.Document;

**import** document.ExcelDocument;

**public** **class** ExcelDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

**return** **new** ExcelDocument();

}

}

PdfDocumentFactory.java:

**package** factory;

**import** document.Document;

**import** document.PdfDocument;

**public** **class** PdfDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

**return** **new** PdfDocument();

}

}

WordDocumentFactory.java:

**package** factory;

**import** document.Document;

**import** document.WordDocument;

**public** **class** WordDocumentFactory **extends** DocumentFactory {

**public** Document createDocument() {

**return** **new** WordDocument();

}

}

factoryPatternTest:

Main.java:

**package** factoryPatternTest;

**import** factory.DocumentFactory;

**import** factory.WordDocumentFactory;

**import** factory.PdfDocumentFactory;

**import** factory.ExcelDocumentFactory;

**import** document.Document;

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

**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();

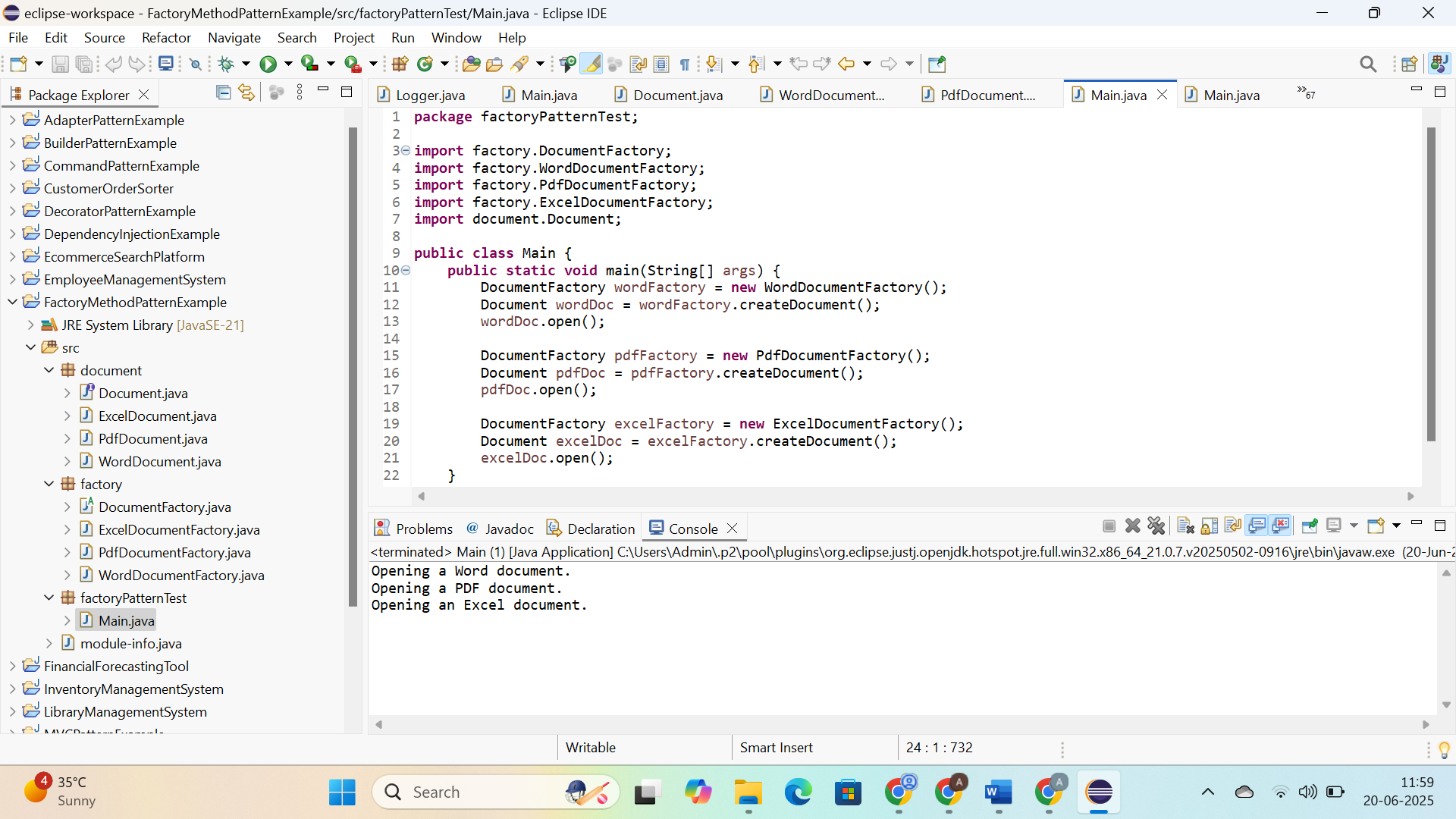
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

OUTPUT:



3: **Implementing the Builder Pattern; Hands On:**

CODE:

model:

Computer.java:

**package** model;

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

**private** **final** String cpu;

**private** **final** String ram;

**private** **final** String storage;

**private** **final** String graphicsCard;

**private** **final** **boolean** isBluetoothEnabled;

**private** Computer(Builder builder) {

**this**.cpu = builder.cpu;

**this**.ram = builder.ram;

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

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

**this**.isBluetoothEnabled = builder.isBluetoothEnabled;

}

**public** String getCpu() { **return** cpu; }

**public** String getRam() { **return** ram; }

**public** String getStorage() { **return** storage; }

**public** String getGraphicsCard() { **return** graphicsCard; }

**public** **boolean** isBluetoothEnabled() { **return** isBluetoothEnabled; }

@Override

**public** String toString() {

**return** "Computer [CPU=" + cpu + ", RAM=" + ram +

", Storage=" + storage + ", GraphicsCard=" + graphicsCard +

", BluetoothEnabled=" + isBluetoothEnabled + "]";

}

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

**private** **final** String cpu;

**private** **final** String ram;

**private** String storage;

**private** String graphicsCard;

**private** **boolean** isBluetoothEnabled;

**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 setBluetoothEnabled(**boolean** isBluetoothEnabled) {

**this**.isBluetoothEnabled = isBluetoothEnabled;

**return** **this**;

}

**public** Computer build() {

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

}

}

}

BuilderPatternExample:

Main.java:

**package** builderPatternExample;

**import** model.Computer;

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

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

Computer basicComputer = **new** Computer.Builder("Intel i5", "8GB")

.build();

Computer gamingComputer = **new** Computer.Builder("AMD Ryzen 9", "32GB")

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 4080")

.setBluetoothEnabled(**true**)

.build();

Computer workComputer = **new** Computer.Builder("Intel i7", "16GB")

.setStorage("512GB SSD")

.setBluetoothEnabled(**true**)

.build();

System.***out***.println("Basic Computer: " + basicComputer);

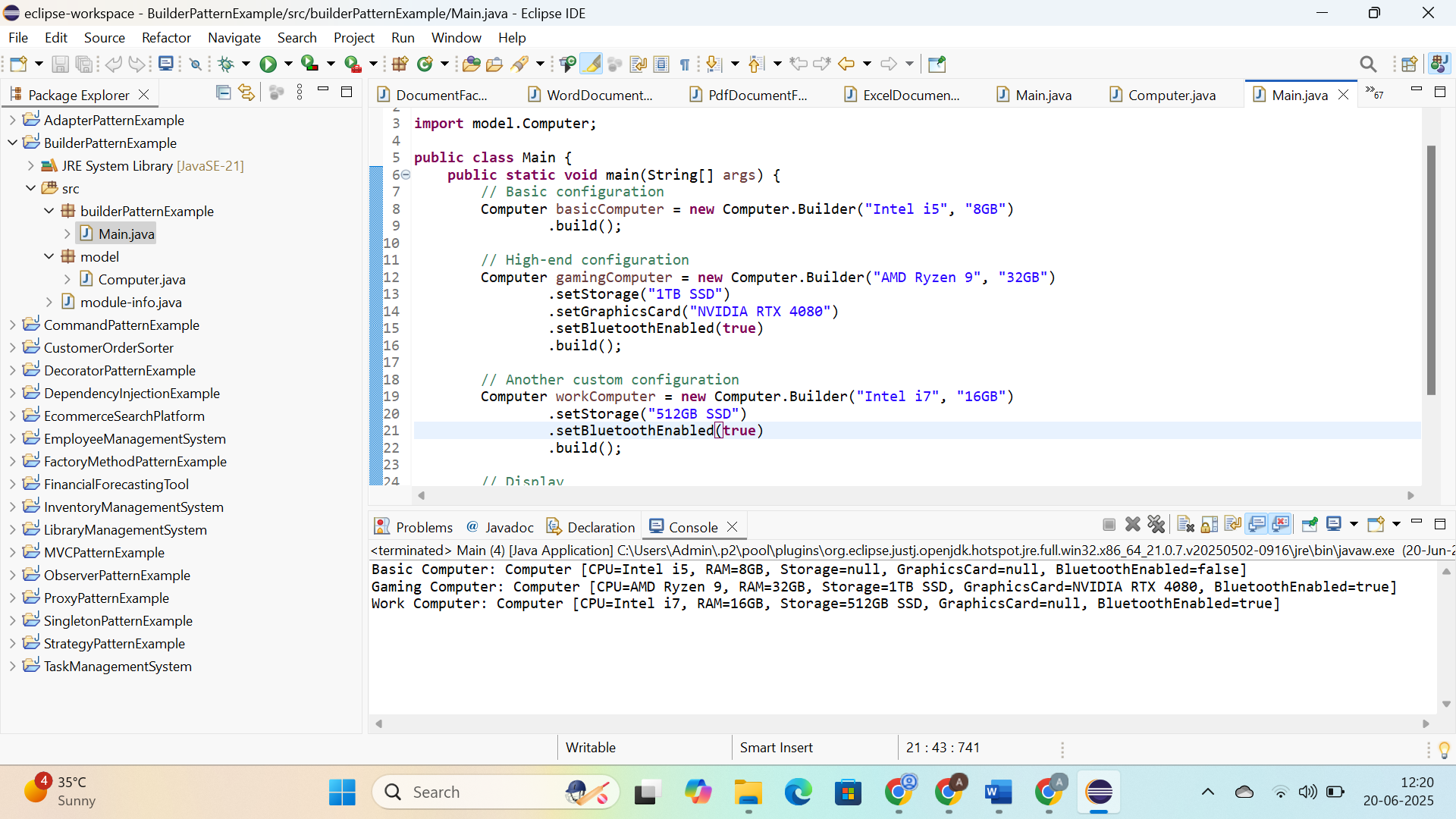
System.***out***.println("Gaming Computer: " + gamingComputer);

System.***out***.println("Work Computer: " + workComputer);

}

}

OUTPUT:



4: **Implementing the Adapter Pattern; Hands On:**

**CODE:**

**adapters:**

**PaypalAdapter.java:**

**package** adapters;

**import** payment.PaymentProcessor;

**import** gateways.PayPalGateway;

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

**private** PayPalGateway paypal;

**public** PayPalAdapter() {

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

}

@Override

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

paypal.sendPayPalPayment(amount);

}

}

**StripeAdapter.java:**

**package** adapters;

**import** payment.PaymentProcessor;

**import** gateways.StripeGateway;

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

**private** StripeGateway stripe;

**public** StripeAdapter() {

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

}

@Override

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

stripe.makeStripePayment(amount);

}

}

**gateways:**

**PayPalGateway.java:**

**package** gateways;

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

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

System.***out***.println("Processing payment with PayPal: $" + amount);

}

}

**StripeGateway.java:**

**package** gateways;

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

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

System.***out***.println("Processing payment with Stripe: $" + amount);

}

}

**payment:**

PaymentProcesser.java:

**package** payment;

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

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

}

adapterPatternExample:

**package** adapterPatternExample;

**import** payment.PaymentProcessor;

**import** adapters.StripeAdapter;

**import** adapters.PayPalAdapter;

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

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

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

stripeProcessor.processPayment(100.00);

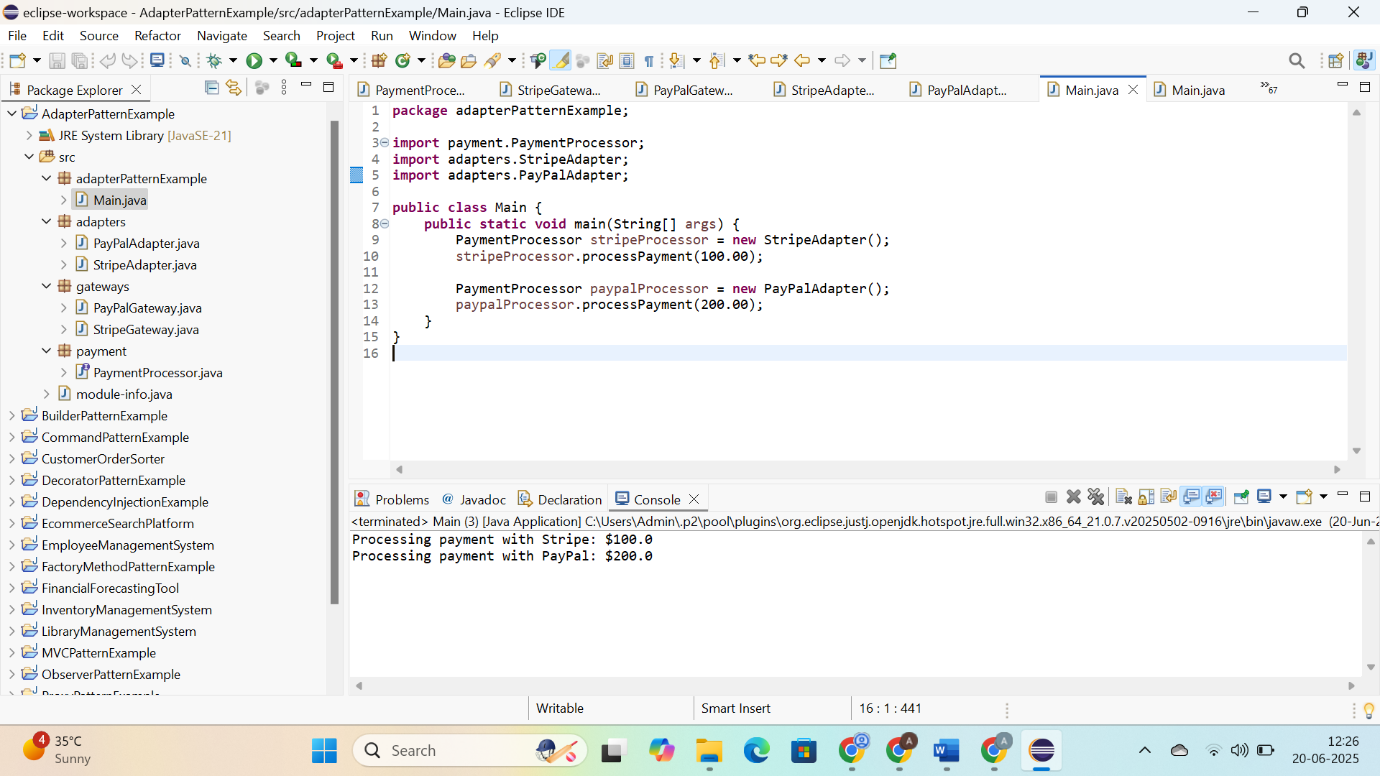
PaymentProcessor paypalProcessor = **new** PayPalAdapter();

paypalProcessor.processPayment(200.00);

}

}

OUTPUT:



5: **Implementing the Decorator Pattern; Hands On:**

**CODE:**

**notifier:**

**EmailNotifier.java:**

**package** notifier;

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

@Override

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

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

}

}

**Notifier.java:**

**package** notifier;

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

**void** send(String message);

}

**NotifierDecorator.java:**

**package** notifier;

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

**protected** Notifier wrappedNotifier;

**public** NotifierDecorator(Notifier notifier) {

**this**.wrappedNotifier = notifier;

}

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

wrappedNotifier.send(message);

}

}

**SlackNotifierDecorator.java:**

**package** notifier;

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

**public** SlackNotifierDecorator(Notifier notifier) {

**super**(notifier);

}

@Override

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

**super**.send(message);

sendSlack(message);

}

**private** **void** sendSlack(String message) {

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

}

}

**SMSNotifierDecorator.java:**

**package** notifier;

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

**public** SMSNotifierDecorator(Notifier notifier) {

**super**(notifier);

}

@Override

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

**super**.send(message);

sendSMS(message);

}

**private** **void** sendSMS(String message) {

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

}

}

**decoratorPatternExample:**

**Main.java:**

**package** decoratorPatternExample;

**import** notifier.\*;

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

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

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

Notifier emailAndSMS = **new** SMSNotifierDecorator(emailNotifier);

Notifier allChannels = **new** SlackNotifierDecorator(emailAndSMS);

System.***out***.println("Sending notification via Email only:");

emailNotifier.send("System will go down at midnight.");

System.***out***.println("\nSending notification via Email + SMS:");

emailAndSMS.send("System will go down at midnight.");

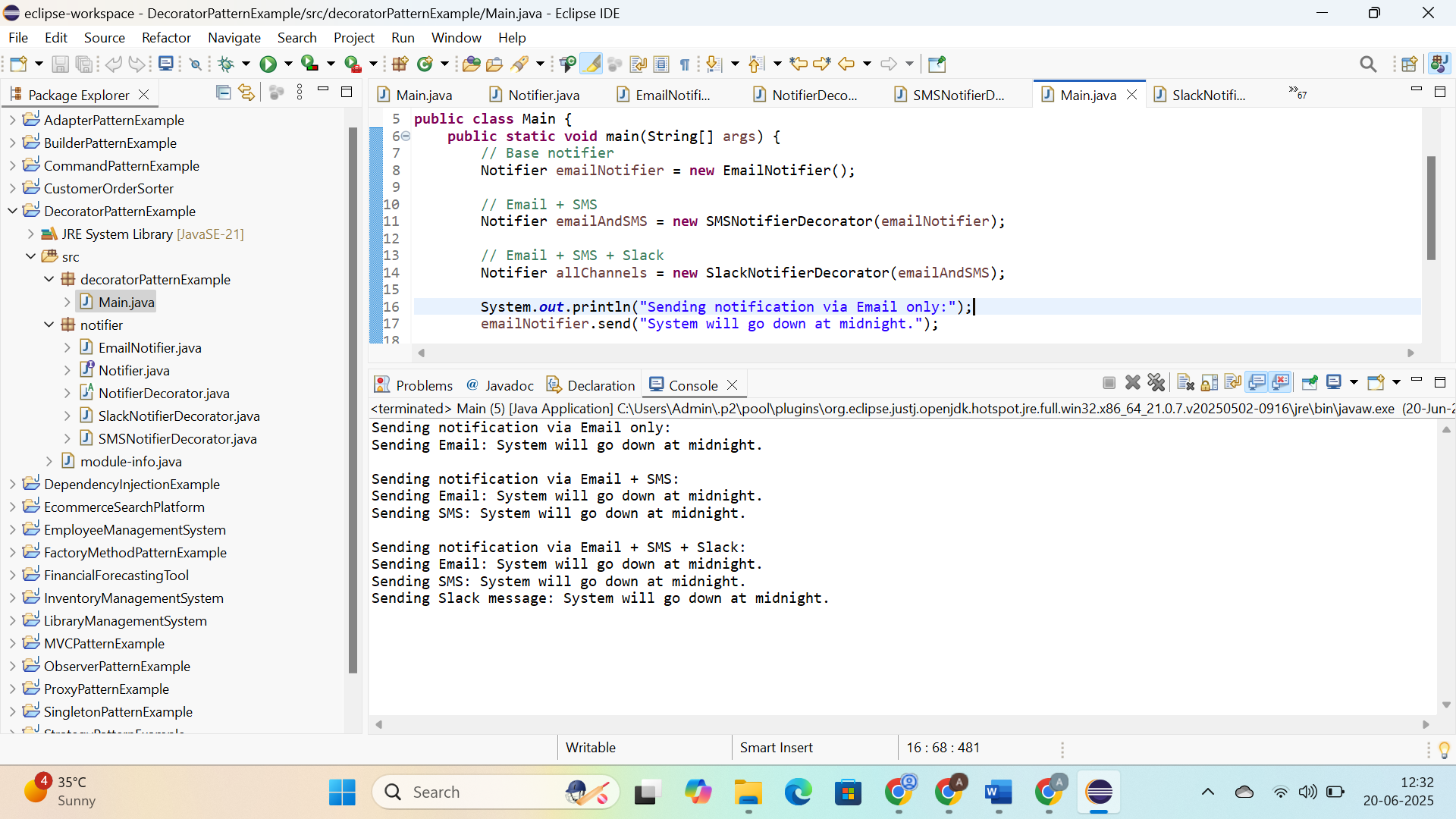
System.***out***.println("\nSending notification via Email + SMS + Slack:");

allChannels.send("System will go down at midnight.");

}

}

OUTPUT:



6: **Implementing the Proxy Pattern; Hands On:**

**CODE:**

**ProxyPatternExample:**

imageviewer:

Image.java:

**package** imageviewer;

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

**void** display();

}

ProxyImage.java:

**package** imageviewer;

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

**private** **final** String filename;

**private** RealImage realImage;

**public** ProxyImage(String filename) {

**this**.filename = filename;

}

@Override

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

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

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

}

realImage.display();

}

}

RealImage.java:

**package** imageviewer;

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

**private** **final** String filename;

**public** RealImage(String filename) {

**this**.filename = filename;

loadFromRemoteServer();

}

**private** **void** loadFromRemoteServer() {

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

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

@Override

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

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

}

}

proxyPatternExample:

Main.java:

**package** proxyPatternExample;

**import** imageviewer.Image;

**import** imageviewer.ProxyImage;

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

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

Image image1 = **new** ProxyImage("image1.jpg");

Image image2 = **new** ProxyImage("image2.jpg");

System.***out***.println("First call to image1:");

image1.display();

System.***out***.println("\nSecond call to image1:");

image1.display();

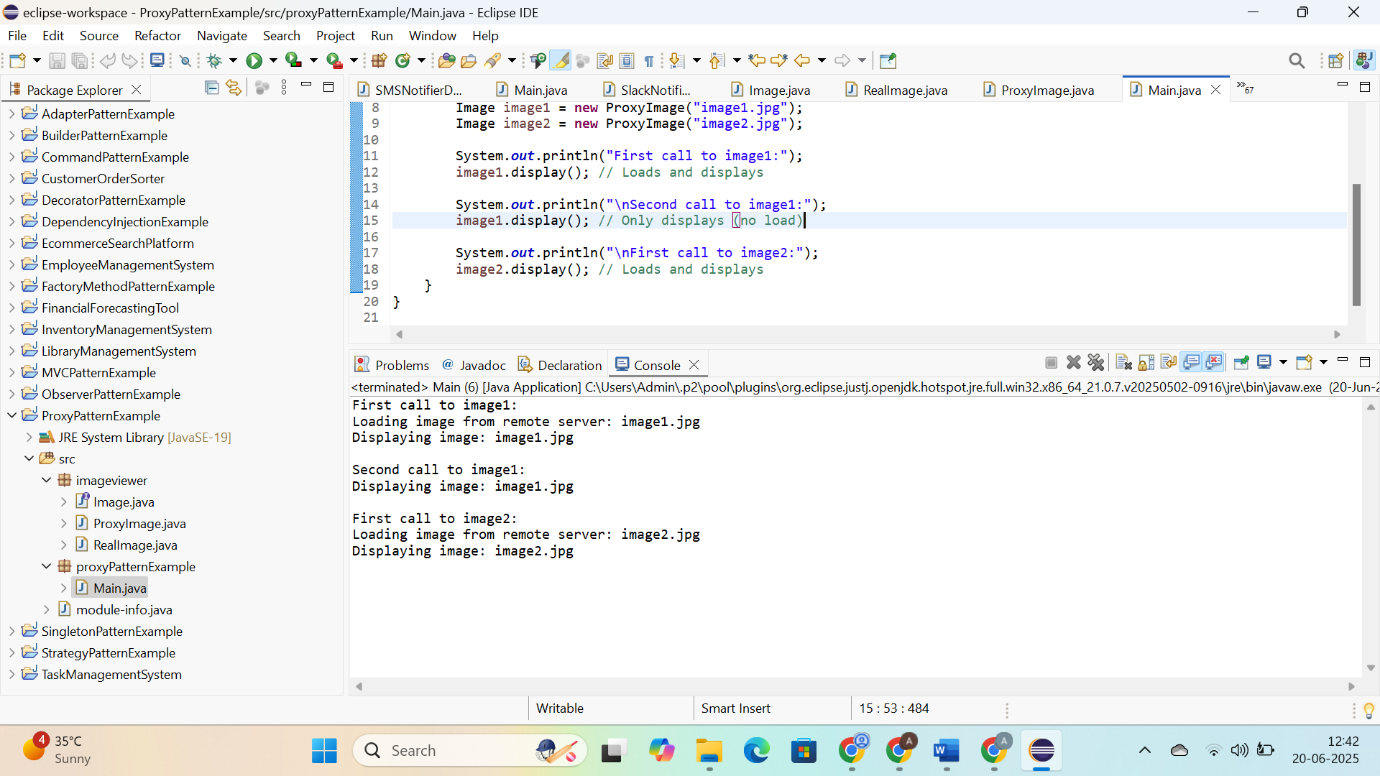
System.***out***.println("\nFirst call to image2:");

image2.display();

}

}

**OUTPUT:**



**7: Implementing the Observer Pattern; Hands On:**

**CODE:**

**observer:**

**MobileApp.java:**

**package** observer;

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

**void** update(**double** price);

}

**Observer.java:**

**package** observer;

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

**private** **final** String name;

**public** MobileApp(String name) {

**this**.name = name;

}

@Override

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

System.***out***.println(name + " (Mobile) received updated stock price: $" + price);

}

}

**WebApp.java:**

**package** observer;

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

**private** **final** String name;

**public** WebApp(String name) {

**this**.name = name;

}

@Override

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

System.***out***.println(name + " (Web) received updated stock price: $" + price);

}

}

**subject:**

**Stock.java:**

**package** subject;

**import** observer.Observer;

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

**void** registerObserver(Observer observer);

**void** removeObserver(Observer observer);

**void** notifyObservers();

}

**StockMarket.java:**

**package** subject;

**import** observer.Observer;

**import** java.util.ArrayList;

**import** java.util.List;

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

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

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

@Override

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

observers.add(observer);

}

@Override

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

observers.remove(observer);

}

@Override

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

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

o.update(stockPrice);

}

}

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

System.***out***.println("\nStock price updated to: $" + price);

**this**.stockPrice = price;

notifyObservers();

}

}

**observerPatternExample:**

**Main.java:**

**package** observerPatternExample;

**import** subject.StockMarket;

**import** observer.MobileApp;

**import** observer.WebApp;

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

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

StockMarket stockMarket = **new** StockMarket();

MobileApp mobileApp1 = **new** MobileApp("Client A");

WebApp webApp1 = **new** WebApp("Client B");

stockMarket.registerObserver(mobileApp1);

stockMarket.registerObserver(webApp1);

stockMarket.setStockPrice(101.50);

stockMarket.setStockPrice(105.25);

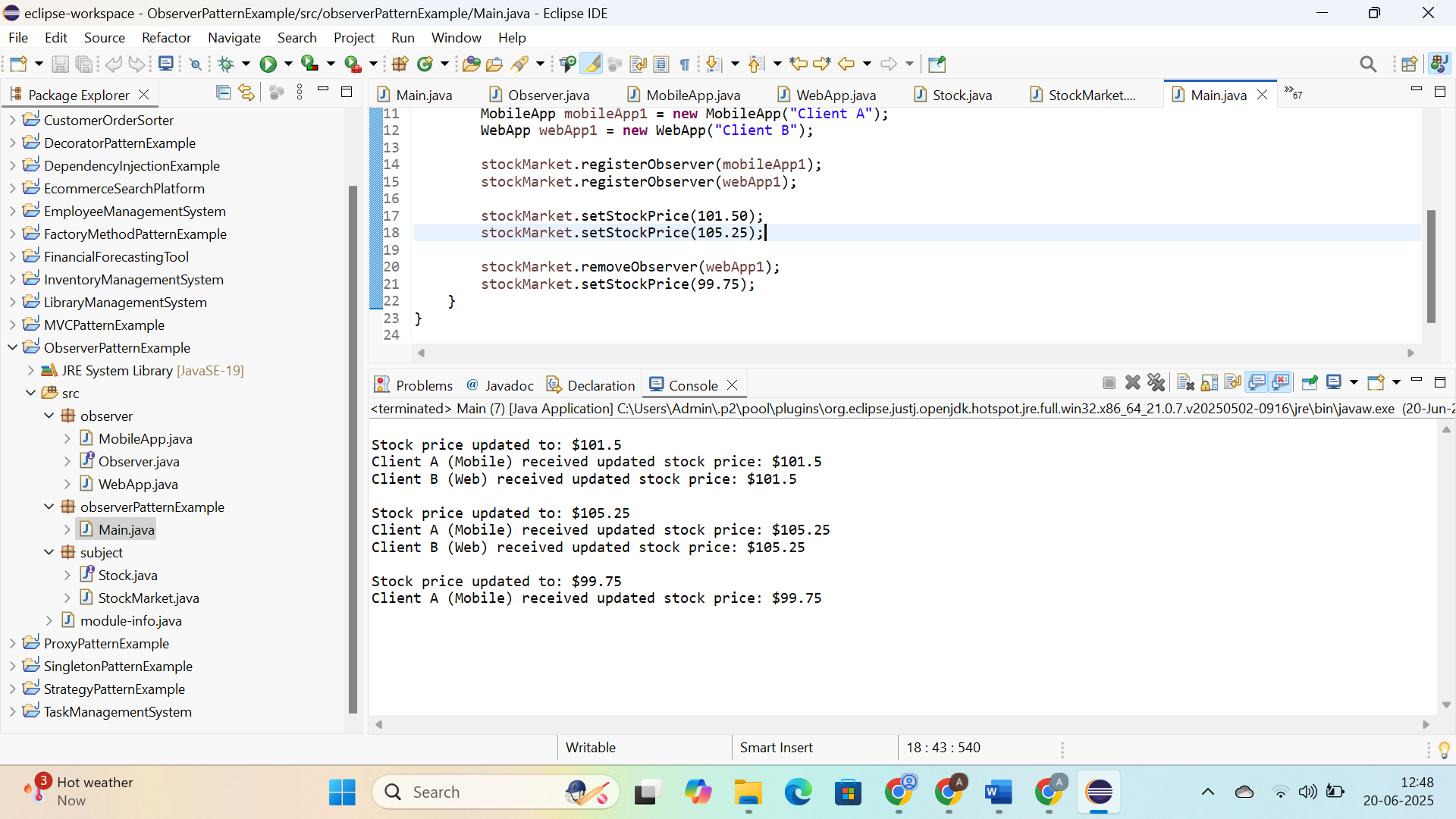
stockMarket.removeObserver(webApp1);

stockMarket.setStockPrice(99.75);

}

}

**OUTPUT:**



**8: Implementing the Strategy Pattern; Hands On:**

**CODE:**

**context:**

**PaymentContext.java:**

**package** context;

**import** strategy.PaymentStrategy;

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

**private** PaymentStrategy strategy;

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

**this**.strategy = strategy;

}

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

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

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

} **else** {

strategy.pay(amount);

}

}

}

**strategy:**

**CreditCardPayment.java:**

**package** strategy;

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

**private** String cardNumber;

**public** CreditCardPayment(String cardNumber) {

**this**.cardNumber = cardNumber;

}

@Override

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

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

}

}

**PaymentStrategy.java:**

**package** strategy;

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

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

}

**PayPalPayment.java:**

**package** strategy;

**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 $" + amount + " using PayPal account: " + email);

}

}

**strategyPatternExample:**

**Main.java:**

**package** strategyPatternExample;

**import** context.PaymentContext;

**import** strategy.CreditCardPayment;

**import** strategy.PayPalPayment;

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

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

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

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

context.pay(250.00);

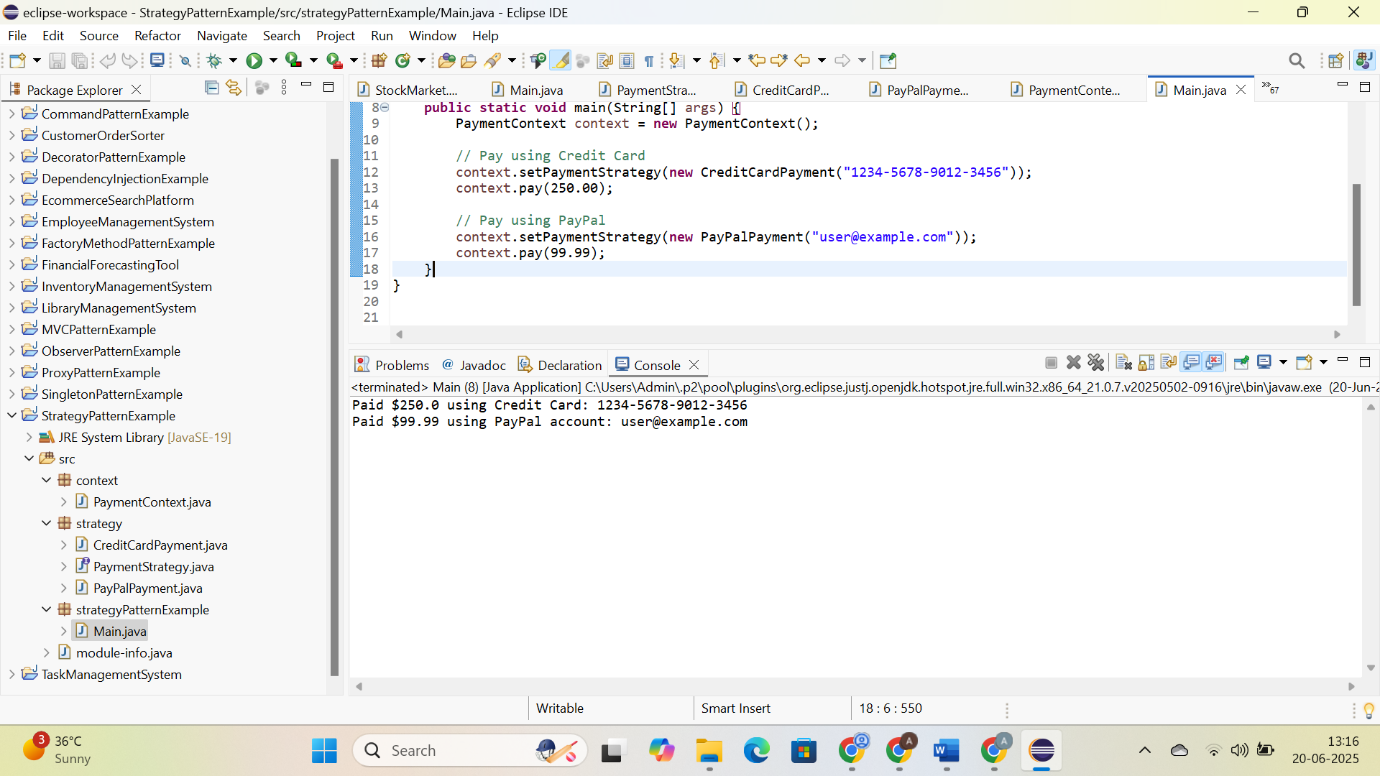
context.setPaymentStrategy(**new** PayPalPayment("user@example.com"));

context.pay(99.99);

}

}

**OUTPUT:**



**9: Implementing the Command Pattern; Hands On:**

**CODE:**

**command:**

**Command.java:**

**package** command;

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

**void** execute();

}

**LightOffCommand.java:**

**package** command;

**import** device.Light;

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

**private** Light light;

**public** LightOffCommand(Light light) {

**this**.light = light;

}

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

light.turnOff();

}

}

**LightOnCommand.java:**

**package** command;

**import** device.Light;

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

**private** Light light;

**public** LightOnCommand(Light light) {

**this**.light = light;

}

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

light.turnOn();

}

}

**device:**

**Light.java:**

**package** device;

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

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

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

}

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

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

}

}

**invoker:**

**RemoteControl.java:**

**package** invoker;

**import** command.Command;

**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.");

}

}

}

**main:**

**CommandPattern:**

**package** main;

**import** command.LightOnCommand;

**import** command.LightOffCommand;

**import** device.Light;

**import** invoker.RemoteControl;

**public** **class** CommandPatternTest {

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

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

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

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

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

remote.setCommand(lightOn);

remote.pressButton();

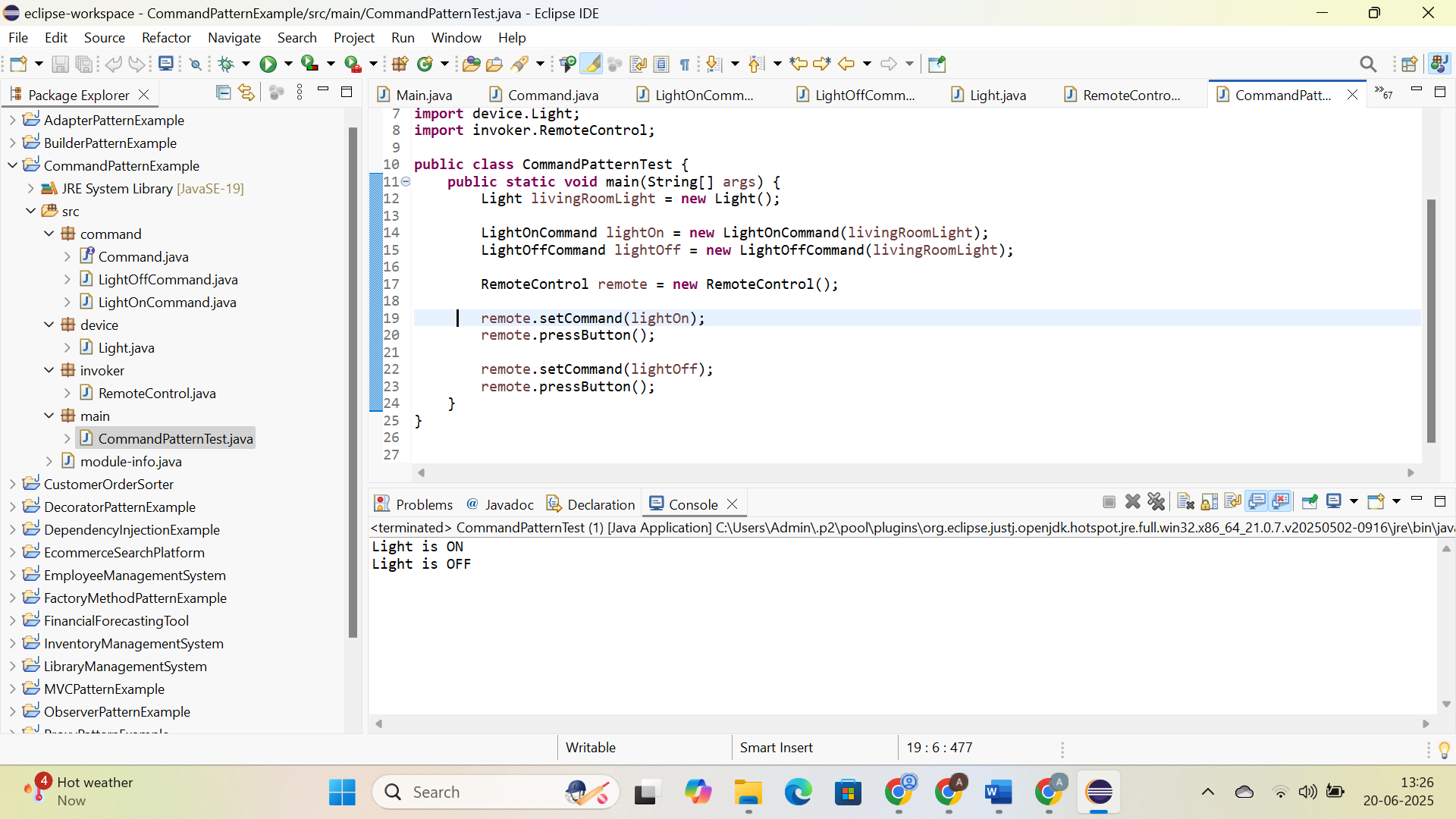
remote.setCommand(lightOff);

remote.pressButton();

}

}

**OUTPUT:**



**10: Implementing the MVC Pattern; Hands On:**

**CODE:**

**controller:**

**StudentController.java:**

**package** controller;

**import** model.Student;

**import** view.StudentView;

**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.getName(), model.getId(), model.getGrade());

}

}

**model:**

**Student.java:**

**package** model;

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

**private** String name;

**private** String id;

**private** String grade;

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

**this**.name = name;

**this**.id = id;

**this**.grade = grade;

}

**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;

}

}

**view:**

**StudentView.java:**

**package** view;

**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);

}

}

**main:**

**MVCTest.java:**

**package** main;

**import** model.Student;

**import** view.StudentView;

**import** controller.StudentController;

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

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

Student student = **new** Student("Alice", "S001", "A");

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

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

controller.updateView();

controller.setStudentName("Bob");

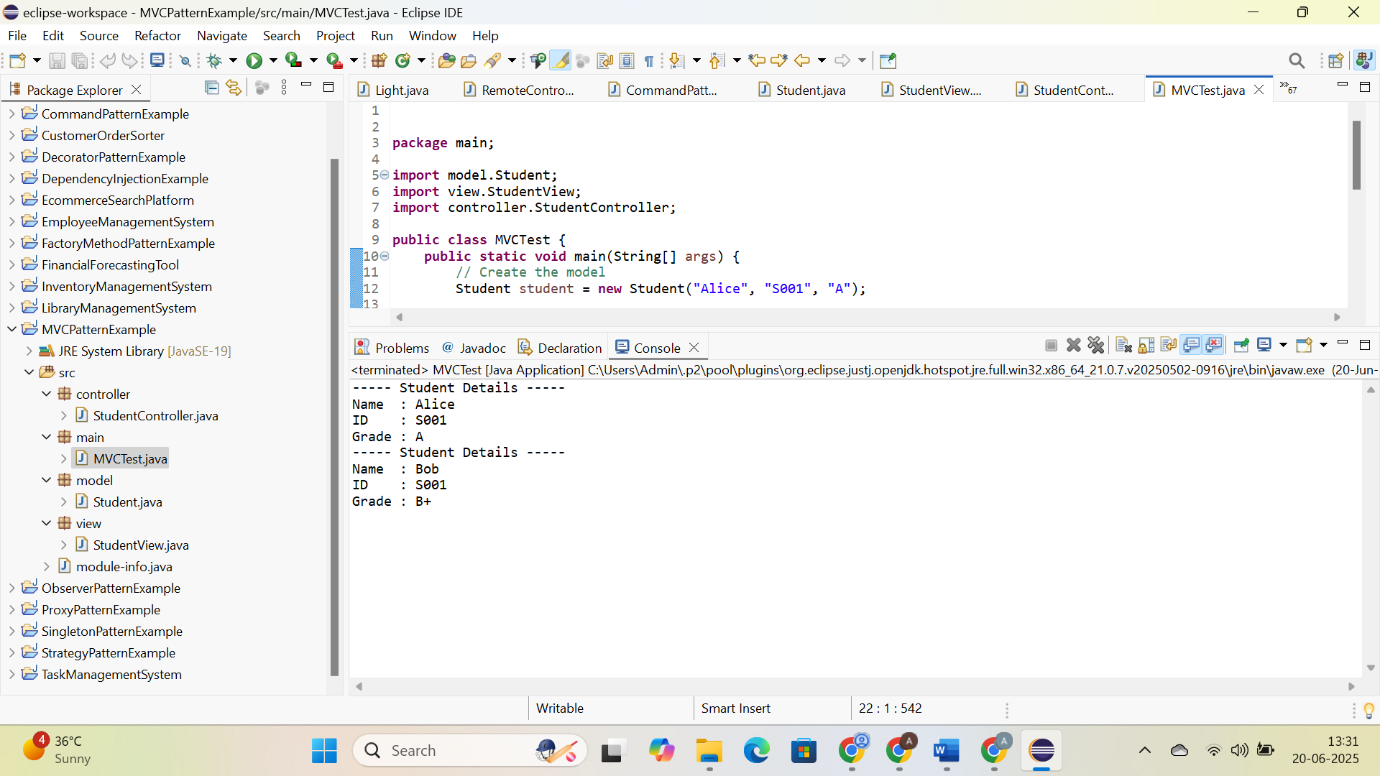
controller.setStudentGrade("B+");

controller.updateView();

}

}

**OUTPUT:**



**11: Implementing Dependency Injection; Hands On:**

**CODE:**

**repository:**

**CustomerRespository.java:**

**package** repository;

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

String findCustomerById(String customerId);

}

**CustomerRepositoryImpl.java:**

**package** repository;

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

@Override

**public** String findCustomerById(String customerId) {

**return** "Customer[ID=" + customerId + ", Name=John Doe]";

}

}

**service:**

**CustomerService.java:**

**package** service;

**import** repository.CustomerRepository;

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

**private** **final** CustomerRepository repository;

**public** CustomerService(CustomerRepository repository) {

**this**.repository = repository;

}

**public** **void** displayCustomer(String customerId) {

String customer = repository.findCustomerById(customerId);

System.***out***.println("Customer Info: " + customer);

}

}

**main:**

**Main.java:**

**package** main;

**import** repository.CustomerRepository;

**import** repository.CustomerRepositoryImpl;

**import** service.CustomerService;

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

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

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

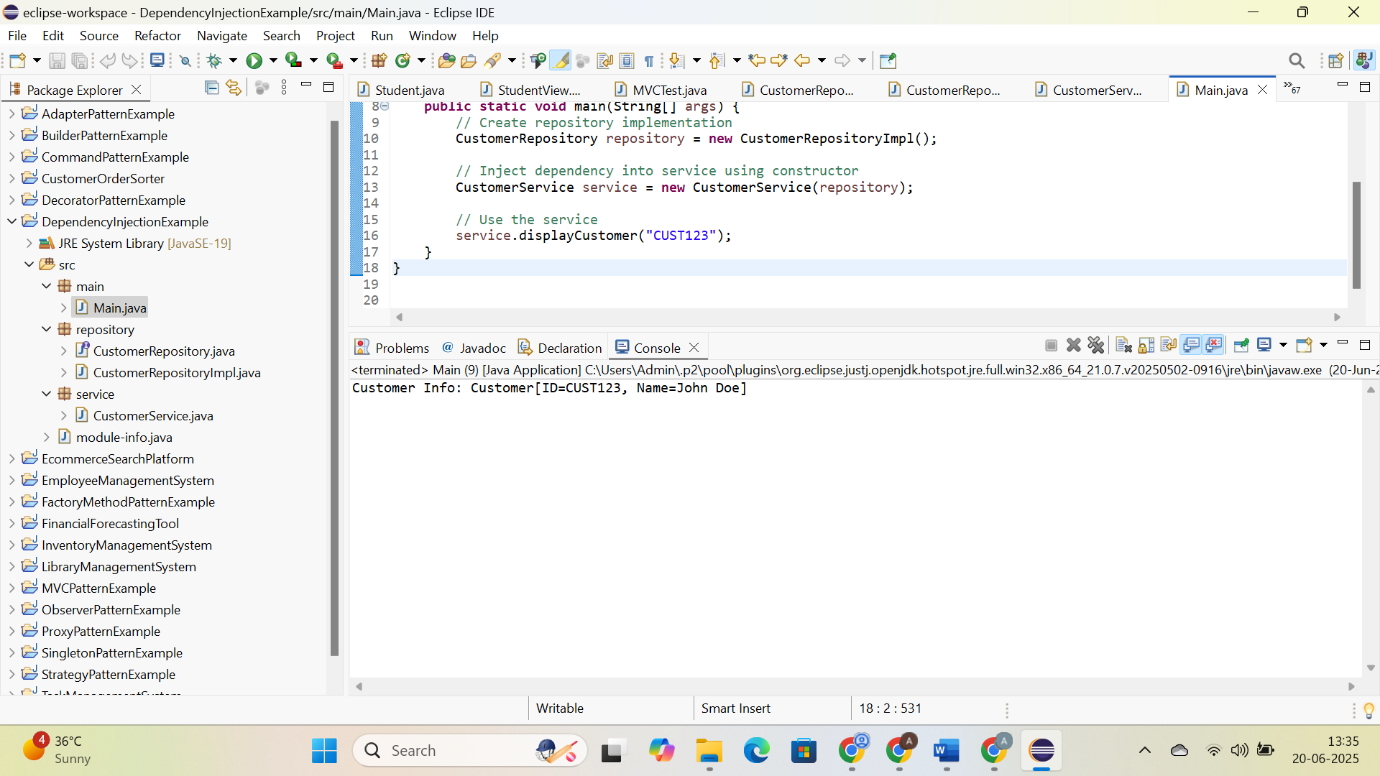
CustomerService service = **new** CustomerService(repository);

service.displayCustomer("CUST123");

}

}

**OUTPUT:**



**ALGORITHMS\_DATA STRUCUTURES**

**1: Inventory Management System; Hands On:**

**CODE:**

**InventoryManagmentSystem:**

**com.inventory.model:**

**Product.java:**

**package** com.inventory.model;

**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** setProductName(String productName) {

**this**.productName = productName;

}

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

**this**.quantity = quantity;

}

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

**this**.price = price;

}

@Override

**public** String toString() {

**return** "Product ID: " + productId + ", Name: " + productName +

", Quantity: " + quantity + ", Price: $" + price;

}

}

**Com.inventory.service:**

**InventoryManager.java:**

**package** com.inventory.service;

**import** com.inventory.model.Product;

**import** java.util.HashMap;

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

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

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

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

}

**public** **void** updateProduct(**int** productId, String name, **int** quantity, **double** price) {

Product product = inventory.get(productId);

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

product.setProductName(name);

product.setQuantity(quantity);

product.setPrice(price);

} **else** {

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

}

}

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

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

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

}

}

**public** Product getProduct(**int** productId) {

**return** inventory.get(productId);

}

**public** **void** displayAllProducts() {

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

System.***out***.println("No products in inventory.");

} **else** {

**for** (Product product : inventory.values()) {

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

}

}

}

}

**com.inventory:**

**Main.java:**

**package** com.inventory;

**import** com.inventory.model.Product;

**import** com.inventory.service.InventoryManager;

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

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

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

manager.addProduct(**new** Product(1, "Laptop", 10, 55000.00));

manager.addProduct(**new** Product(2, "Monitor", 20, 12000.50));

manager.addProduct(**new** Product(3, "Mouse", 100, 450.75));

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

manager.displayAllProducts();

manager.updateProduct(2, "LED Monitor", 15, 11500.00);

manager.deleteProduct(3);

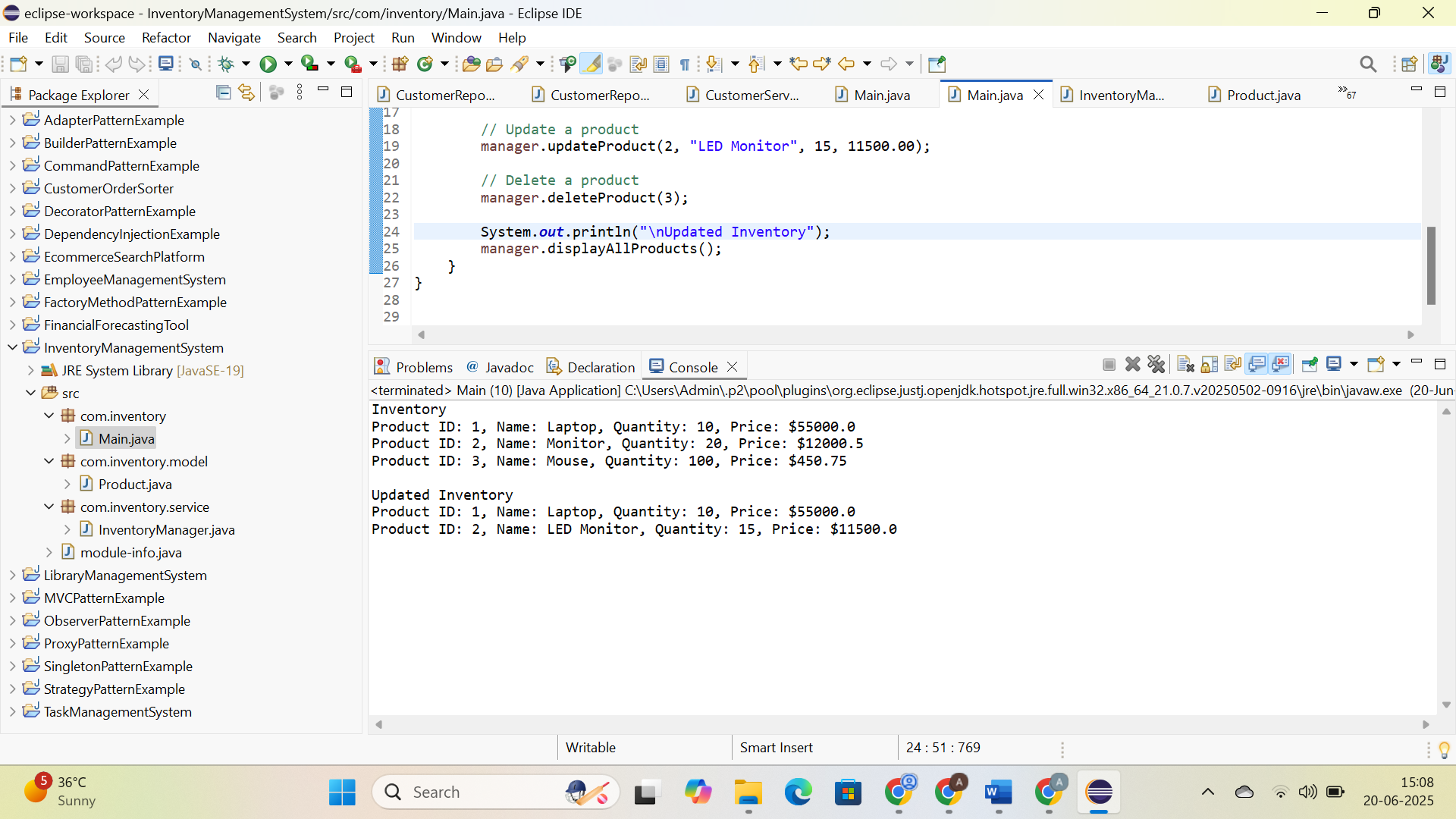
System.***out***.println("\nUpdated Inventory");

manager.displayAllProducts();

}

}

**OUTPUT:**



**2: E-commerce Platform Search Function; Hands On:**

**CODE:**

**EcommerceSearchPlatform:**

**com.ecommerce.model:**

**Product.java:**

**package** com.ecommerce.model;

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

**return** productId;

}

**public** String getProductName() {

**return** productName;

}

**public** String getCategory() {

**return** category;

}

@Override

**public** String toString() {

**return** "Product ID: " + productId +

", Name: " + productName +

", Category: " + category;

}

}

**com.ecommerce.service:**

**SearchService.java:**

**package** com.ecommerce.service;

**import** com.ecommerce.model.Product;

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** SearchService {

**public** **static** Product linearSearch(Product[] products, String targetName) {

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

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

**return** product;

}

}

**return** **null**;

}

**public** **static** Product binarySearch(Product[] products, String targetName) {

**int** low = 0;

**int** high = products.length - 1;

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

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

**int** comparison = products[mid].getProductName().compareToIgnoreCase(targetName);

**if** (comparison == 0) {

**return** products[mid];

} **else** **if** (comparison < 0) {

low = mid + 1;

} **else** {

high = mid - 1;

}

}

**return** **null**;

}

**public** **static** **void** sortByProductName(Product[] products) {

Arrays.*sort*(products, Comparator.*comparing*(Product::getProductName, String.***CASE\_INSENSITIVE\_ORDER***));

}

}

**com.ecommerce:**

**Main.java:**

**package** com.ecommerce;

**import** com.ecommerce.model.Product;

**import** com.ecommerce.service.SearchService;

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

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

Product[] products = {

**new** Product(101, "Shoes", "Footwear"),

**new** Product(102, "Laptop", "Electronics"),

**new** Product(103, "Watch", "Accessories"),

**new** Product(104, "Camera", "Electronics"),

**new** Product(105, "T-Shirt", "Clothing")

};

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

Product resultLinear = SearchService.*linearSearch*(products, "Watch");

System.***out***.println(resultLinear != **null** ? resultLinear : "Product not found.");

SearchService.*sortByProductName*(products);

System.***out***.println("\nBinary Search for 'Watch':");

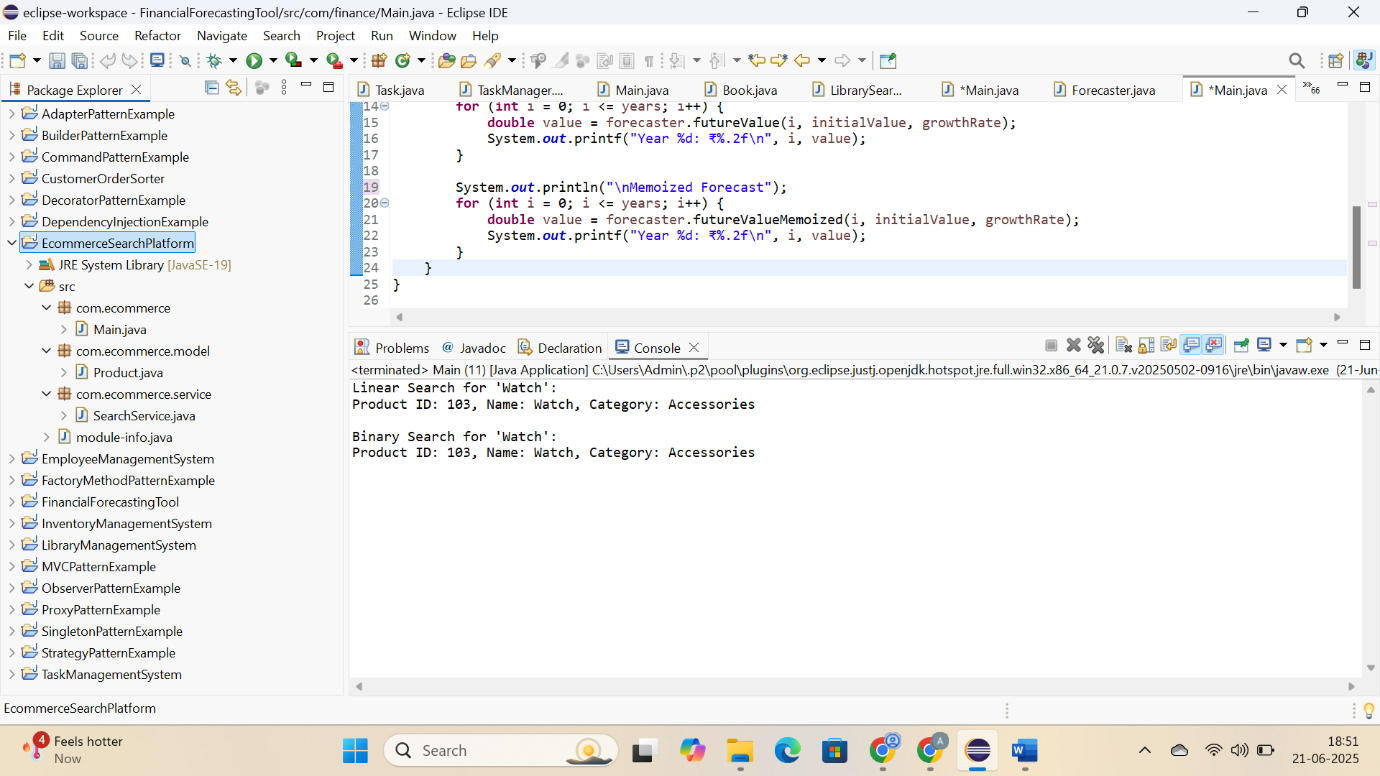
Product resultBinary = SearchService.*binarySearch*(products, "Watch");

System.***out***.println(resultBinary != **null** ? resultBinary : "Product not found.");

}

}

**OUTPUT:**



**3: Sorting Customer Orders; Hands On:**

**CODE:**

**CustomerOrderSorter:**

**com.ecommerce.model:**

**Order.java:**

**package** com.ecommerce.model;

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

**private** **int** orderId;

**private** String customerName;

**private** **double** totalPrice;

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

**this**.orderId = orderId;

**this**.customerName = customerName;

**this**.totalPrice = totalPrice;

}

**public** **int** getOrderId() {

**return** orderId;

}

**public** String getCustomerName() {

**return** customerName;

}

**public** **double** getTotalPrice() {

**return** totalPrice;

}

@Override

**public** String toString() {

**return** "Order ID: " + orderId + ", Customer: " + customerName + ", Total Price: $" + totalPrice;

}

}

**com.ecommerce.service:**

**OrderSorter.java:**

**package** com.ecommerce.service;

**import** com.ecommerce.model.Order;

**public** **class** OrderSorter {

**public** **static** **void** bubbleSort(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].getTotalPrice() > orders[j + 1].getTotalPrice()) {

Order temp = orders[j];

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

orders[j + 1] = temp;

}

}

}

}

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

**if** (low < high) {

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

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

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

}

}

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

**double** pivot = orders[high].getTotalPrice();

**int** i = low - 1;

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

**if** (orders[j].getTotalPrice() < 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;

}

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

**for** (Order order : orders) {

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

}

}

}

**com.ecommerce:**

**Main.java:**

**package** com.ecommerce;

**import** com.ecommerce.model.Order;

**import** com.ecommerce.service.OrderSorter;

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

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

Order[] orders1 = {

**new** Order(201, "Alice", 999.99),

**new** Order(202, "Bob", 499.50),

**new** Order(203, "Charlie", 750.00),

**new** Order(204, "David", 1200.00)

};

Order[] orders2 = orders1.clone();

System.***out***.println(" Before Sorting ");

OrderSorter.*displayOrders*(orders1);

System.***out***.println("\n Bubble Sort by Total Price");

OrderSorter.*bubbleSort*(orders1);

OrderSorter.*displayOrders*(orders1);

System.***out***.println("\nQuick Sort by Total Price");

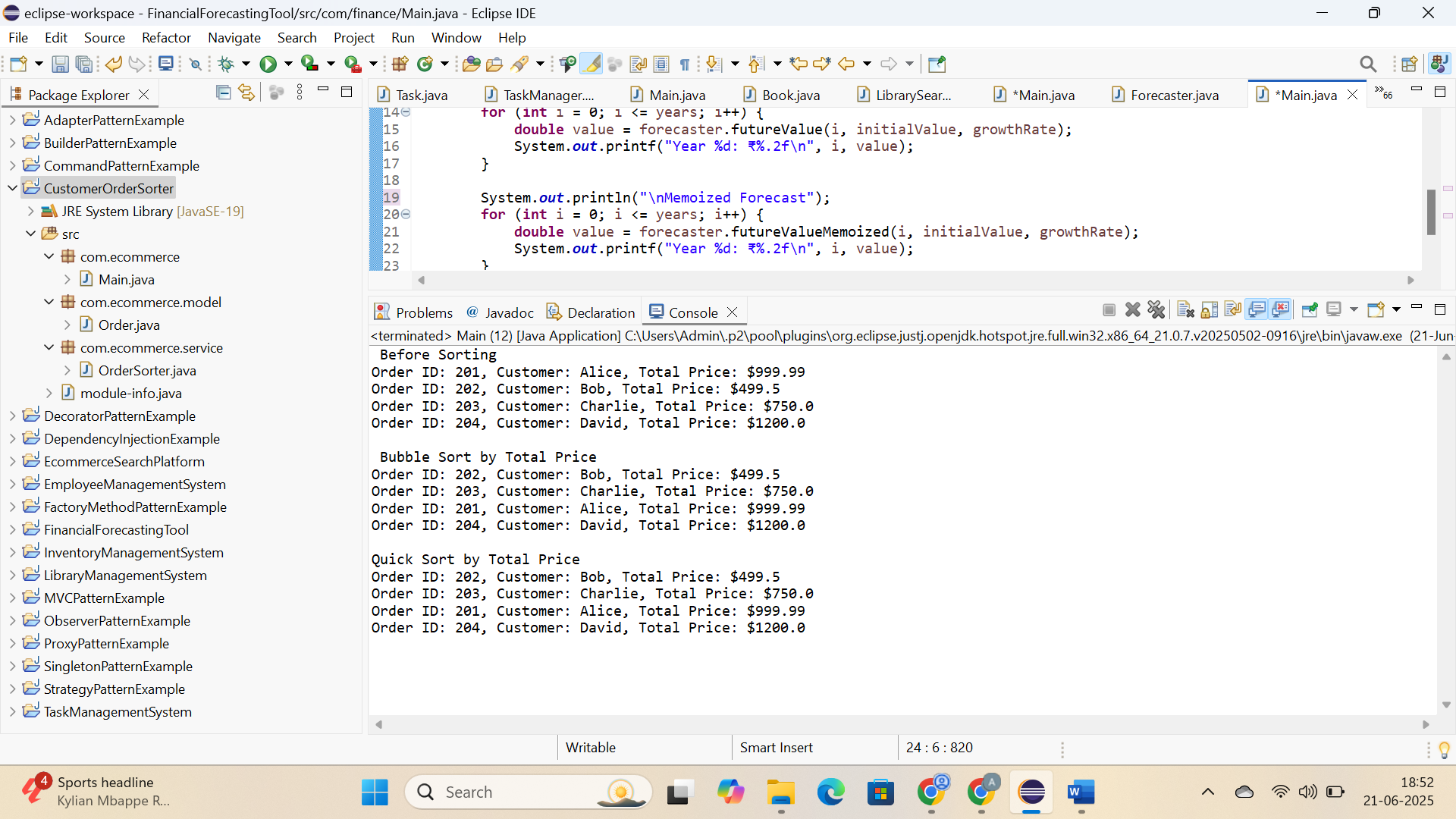
OrderSorter.*quickSort*(orders2, 0, orders2.length - 1);

OrderSorter.*displayOrders*(orders2);

}

}

**OUTPUT:**



**4: Employee Management System**

**CODE:**

**EmployeeManagementSystem:**

**com.company.model:**

**Employee.java:**

**package** com.company.model;

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

**private** **int** employeeId;

**private** String name;

**private** String position;

**private** **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** **int** getEmployeeId() { **return** employeeId; }

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

**public** String getPosition() { **return** position; }

**public** **double** getSalary() { **return** salary; }

@Override

**public** String toString() {

**return** "ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: ₹" + salary;

}

}

**com.company.service:**

**EmployeeManager.java:**

**package** com.company.service;

**import** com.company.model.Employee;

**public** **class** EmployeeManager {

**private** Employee[] employees;

**private** **int** count;

**public** EmployeeManager(**int** size) {

employees = **new** Employee[size];

count = 0;

}

**public** **void** addEmployee(Employee emp) {

**if** (count < employees.length) {

employees[count++] = emp;

} **else** {

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

}

}

**public** Employee searchEmployee(**int** empId) {

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

**if** (employees[i].getEmployeeId() == empId) {

**return** employees[i];

}

}

**return** **null**;

}

**public** **void** listEmployees() {

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

System.***out***.println("No employees found.");

**return**;

}

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

System.***out***.println(employees[i]);

}

}

**public** **void** deleteEmployee(**int** empId) {

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

**if** (employees[i].getEmployeeId() == empId) {

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

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

}

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

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

**return**;

}

}

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

}

}

**com.company:**

**Main.java:**

**package** com.company;

**import** com.company.model.Employee;

**import** com.company.service.EmployeeManager;

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

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

EmployeeManager manager = **new** EmployeeManager(5); // array size 5

manager.addEmployee(**new** Employee(1, "Anil", "Manager", 85000));

manager.addEmployee(**new** Employee(2, "Sneha", "Developer", 70000));

manager.addEmployee(**new** Employee(3, "Ravi", "Designer", 60000));

System.***out***.println("All Employees:");

manager.listEmployees();

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

Employee emp = manager.searchEmployee(2);

System.***out***.println(emp != **null** ? emp : "Not found");

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

manager.deleteEmployee(1);

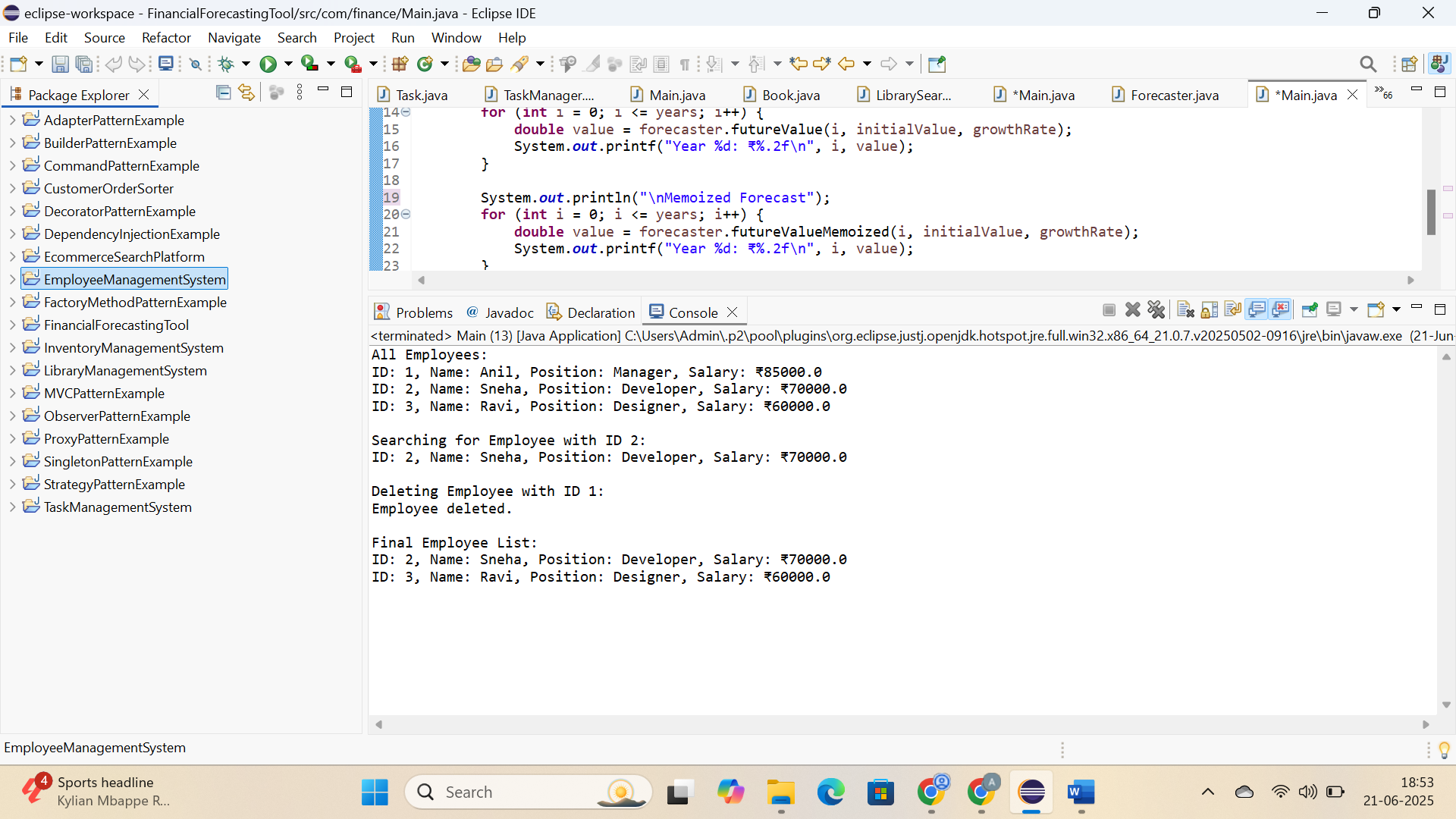
System.***out***.println("\nFinal Employee List:");

manager.listEmployees();

}

}

**OUTPUT:**



**5: Task Management System; Hands On:**

**CODE:**

**TaskManagementSystem:**

**com.tasks.model:**

**Task.java:**

**package** com.tasks.model;

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

**private** **int** taskId;

**private** String taskName;

**private** String status;

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

**this**.taskId = taskId;

**this**.taskName = taskName;

**this**.status = status;

}

**public** **int** getTaskId() { **return** taskId; }

**public** String getTaskName() { **return** taskName; }

**public** String getStatus() { **return** status; }

@Override

**public** String toString() {

**return** "Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status;

}

}

**com.tasks.service**

**TaskManager.java:**

**package** com.tasks.service;

**import** com.tasks.model.Task;

**public** **class** TaskManager {

**private** **class** Node {

Task task;

Node next;

Node prev;

Node(Task task) {

**this**.task = task;

}

}

**private** Node head;

**private** Node tail;

**public** **void** addTask(Task task) {

Node newNode = **new** Node(task);

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

head = tail = newNode;

} **else** {

tail.next = newNode;

newNode.prev = tail;

tail = newNode;

}

}

**public** Task searchTask(**int** taskId) {

Node current = head;

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

**if** (current.task.getTaskId() == taskId) {

**return** current.task;

}

current = current.next;

}

**return** **null**;

}

**public** **void** deleteTask(**int** taskId) {

Node current = head;

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

**if** (current.task.getTaskId() == taskId) {

**if** (current.prev != **null**) {

current.prev.next = current.next;

} **else** {

head = current.next;

}

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

current.next.prev = current.prev;

} **else** {

tail = current.prev;

}

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

**return**;

}

current = current.next;

}

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

}

**public** **void** displayTasksForward() {

Node current = head;

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

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

**return**;

}

System.***out***.println("Task List (Forward)");

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

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

current = current.next;

}

}

**public** **void** displayTasksBackward() {

Node current = tail;

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

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

**return**;

}

System.***out***.println(" Task List (Backward)");

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

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

current = current.prev;

}

}

}

**com.task:**

**Main.java:**

**package** com.tasks;

**import** com.tasks.model.Task;

**import** com.tasks.service.TaskManager;

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

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

TaskManager manager = **new** TaskManager();

manager.addTask(**new** Task(1, "Write Requirements", "Pending"));

manager.addTask(**new** Task(2, "Code Feature", "In Progress"));

manager.addTask(**new** Task(3, "Test Module", "Not Started"));

manager.displayTasksForward();

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

System.***out***.println(manager.searchTask(2));

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

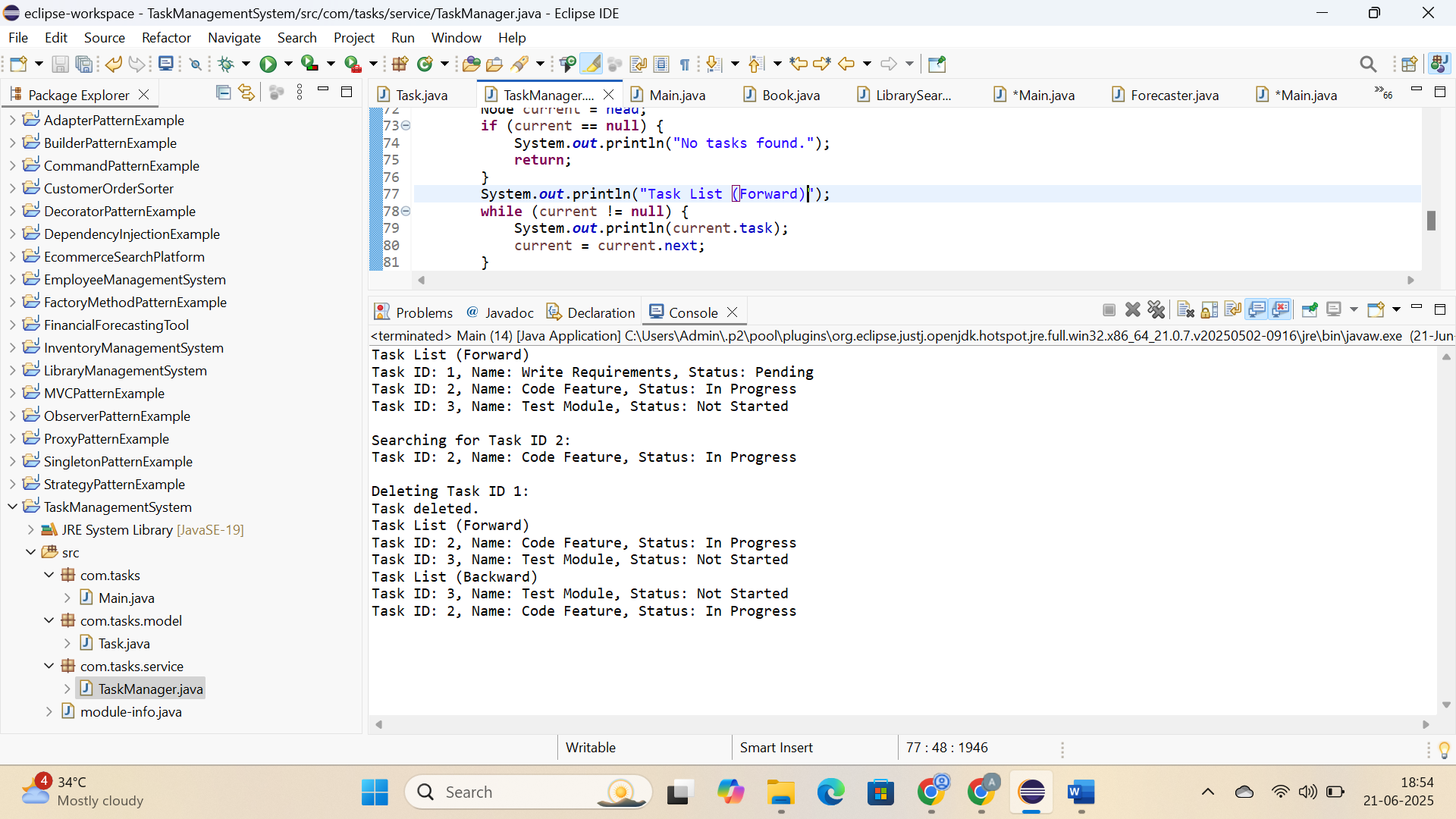
manager.deleteTask(1);

manager.displayTasksForward();

manager.displayTasksBackward();

}

}

**OUTPUT:**

**6: Library Management System; Hands On:**

**CODE:**

**LibraryManagementSystem:**

**com.library.model:**

**Book.java:**

**package** com.library.model;

**public** **class** Book {

**private** **int** bookId;

**private** String title;

**private** String author;

**public** Book(**int** bookId, String title, String author) {

**this**.bookId = bookId;

**this**.title = title;

**this**.author = author;

}

**public** **int** getBookId() { **return** bookId; }

**public** String getTitle() { **return** title; }

**public** String getAuthor() { **return** author; }

@Override

**public** String toString() {

**return** "Book ID: " + bookId + ", Title: \"" + title + "\", Author: " + author;

}

}

**com.library.service:**

**LibrarySearch.java:**

**package** com.library.service;

**import** com.library.model.Book;

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** LibrarySearch {

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

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

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

**return** book;

}

}

**return** **null**;

}

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

**int** low = 0;

**int** high = books.length - 1;

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

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

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

**if** (compare == 0) {

**return** books[mid];

} **else** **if** (compare < 0) {

low = mid + 1;

} **else** {

high = mid - 1;

}

}

**return** **null**;

}

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

Arrays.*sort*(books, Comparator.*comparing*(Book::getTitle, String.***CASE\_INSENSITIVE\_ORDER***));

}

}

**com.library:**

**Main.java:**

**package** com.library;

**import** com.library.model.Book;

**import** com.library.service.LibrarySearch;

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

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

Book[] books = {

**new** Book(101, "Java Programming", "James Gosling"),

**new** Book(102, "Python Basics", "Guido van Rossum"),

**new** Book(103, "C++ Primer", "Bjarne Stroustrup"),

**new** Book(104, "Data Structures", "Mark Allen Weiss"),

**new** Book(105, "Algorithms", "Robert Sedgewick")

};

System.***out***.println("Linear Search: 'Python Basics'");

Book result1 = LibrarySearch.*linearSearch*(books, "Python Basics");

System.***out***.println(result1 != **null** ? result1 : "Book not found.");

LibrarySearch.*sortBooksByTitle*(books);

System.***out***.println("\nBinary Search: 'Python Basics'");

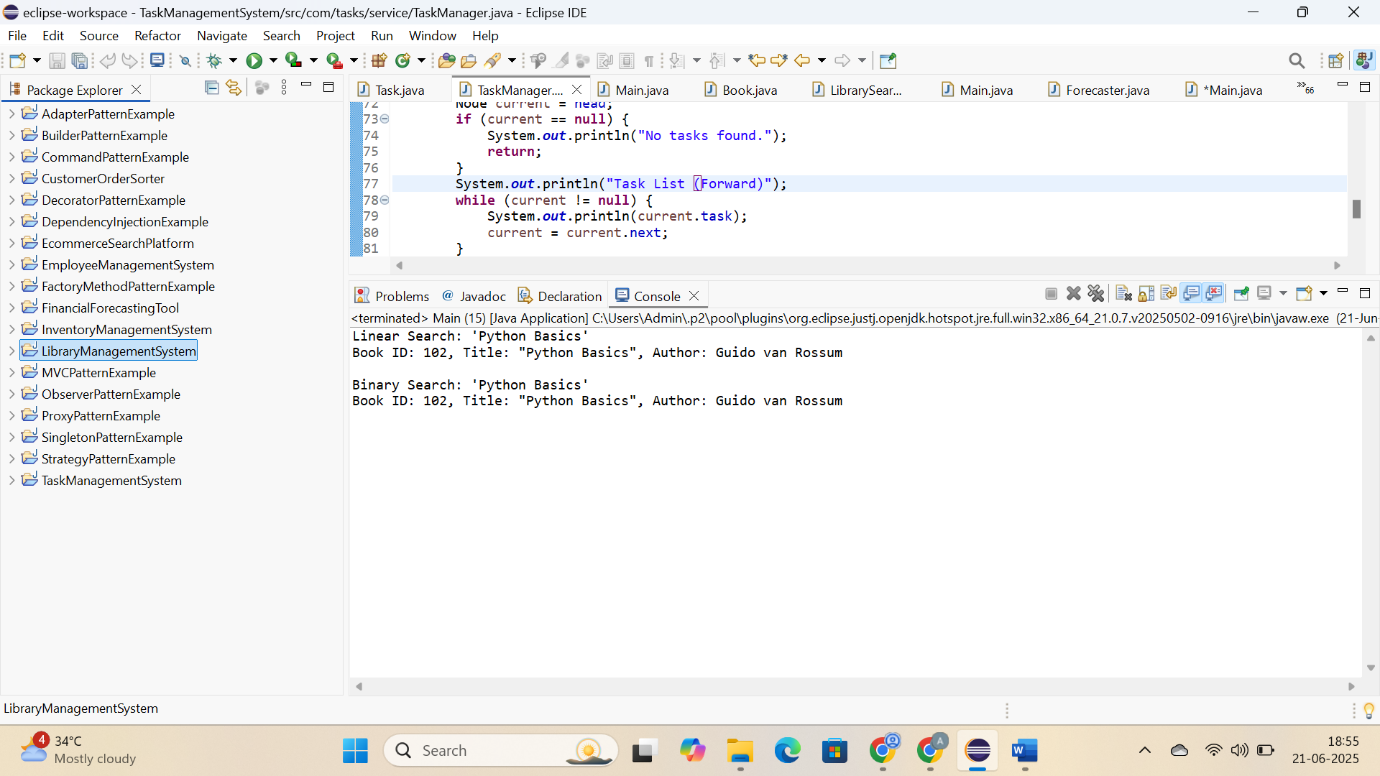
Book result2 = LibrarySearch.*binarySearch*(books, "Python Basics");

System.***out***.println(result2 != **null** ? result2 : "Book not found.");

}

}

**OUTPUT:**



**7: Financial Forecasting; Hands On:**

**CODE:**

**FinancialForecastingTool:**

**com.finance.service:**

**Forecaster.java:**

**package** com.finance.service;

**import** java.util.HashMap;

**public** **class** Forecaster {

**public** **double** futureValue(**int** year, **double** initialValue, **double** rate) {

**if** (year == 0) {

**return** initialValue;

}

**return** futureValue(year - 1, initialValue, rate) \* (1 + rate);

}

**private** HashMap<Integer, Double> memo = **new** HashMap<>();

**public** **double** futureValueMemoized(**int** year, **double** initialValue, **double** rate) {

**if** (year == 0) {

**return** initialValue;

}

**if** (memo.containsKey(year)) {

**return** memo.get(year);

}

**double** result = futureValueMemoized(year - 1, initialValue, rate) \* (1 + rate);

memo.put(year, result);

**return** result;

}

}

**com.finance:**

**Main.java:**

**package** com.finance;

**import** com.finance.service.Forecaster;

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

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

Forecaster forecaster = **new** Forecaster();

**double** initialValue = 10000;

**double** growthRate = 0.10;

**int** years = 5;

System.***out***.println("Recursive Forecast");

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

**double** value = forecaster.futureValue(i, initialValue, growthRate);

System.***out***.printf("Year %d: ₹%.2f\n", i, value);

}

System.***out***.println("\nMemoized Forecast");

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

**double** value = forecaster.futureValueMemoized(i, initialValue, growthRate);

System.***out***.printf("Year %d: ₹%.2f\n", i, value);

}

}

}

**OUTPUT:**

