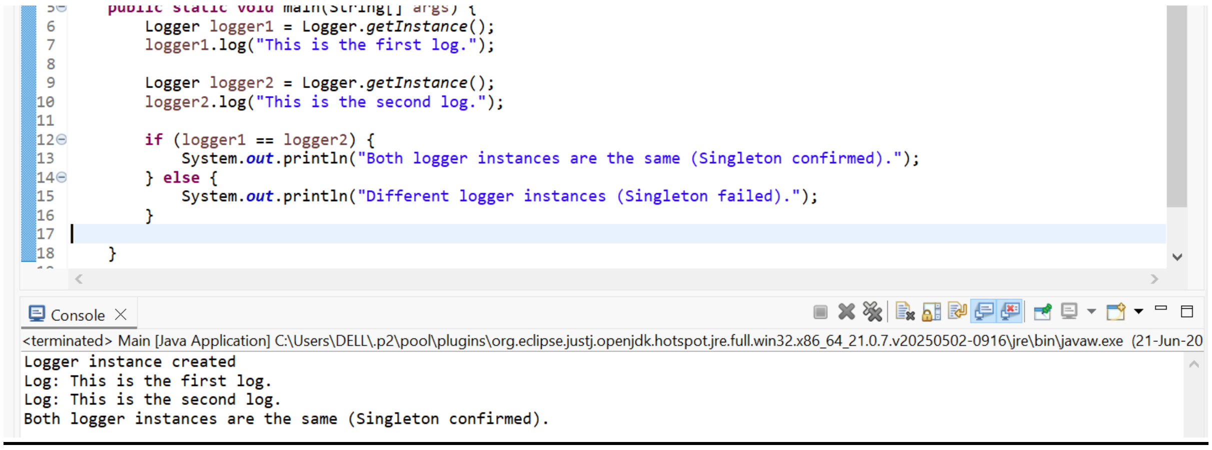
STUDENT NAME: VIDYA CS **TRACK: JAVA FSE**

Exercise 1: Implementing the Singleton Pattern

```
package singleton;
public class Logger {
  //private instance logger
  private static Logger instance;
  //make the constructor private
  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);
  }
}
public class Main {
        public static void main(String[] args) {
                Logger logger1 = Logger.getInstance();
    logger1.log("This is the first log.");
    Logger logger2 = Logger.getInstance();
    logger2.log("This is the second log.");
    if (logger1 == logger2) {
      System.out.println("Both logger instances are the same (Singleton confirmed).");
    } else {
            System.out.println("Different logger instances (Singleton failed).");
        }
            }
}
```

OUTPUT:



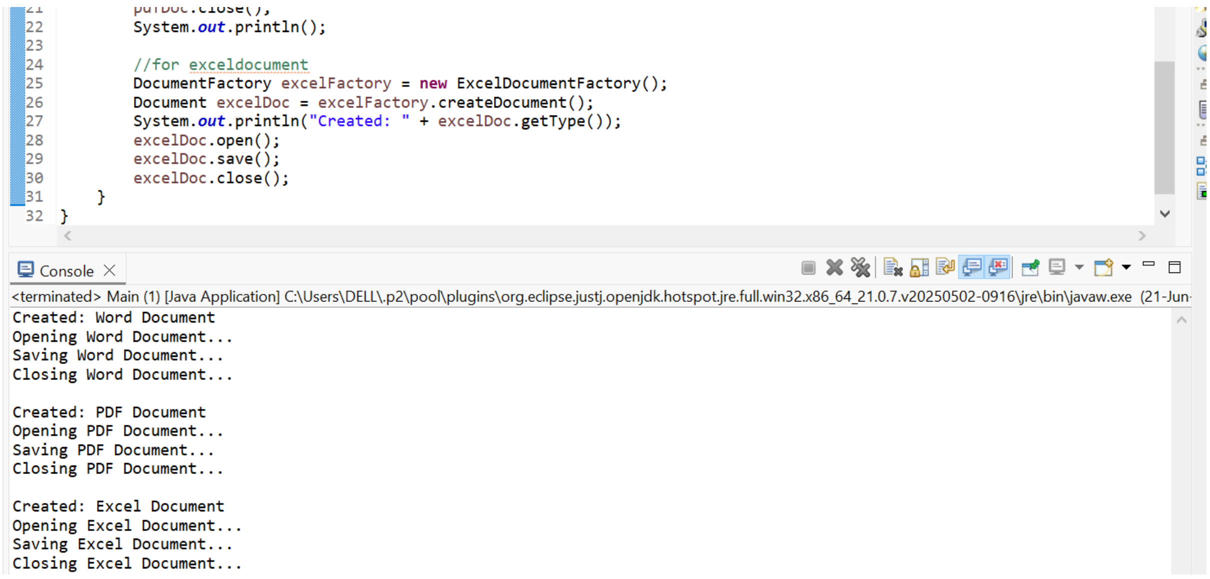
```
    public static void main(String[] args) {
        Logger logger1 = Logger.getInstance();
        logger1.log("This is the first log.");

        Logger logger2 = Logger.getInstance();
        logger2.log("This is the second log.");

        if (logger1 == logger2) {
            System.out.println("Both logger instances are the same (Singleton confirmed).");
        } else {
            System.out.println("Different logger instances (Singleton failed).");
        }

    }
```

```
<terminated> Main [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe  (21-Jun-20
Logger instance created
Log: This is the first log.
Log: This is the second log.
Both logger instances are the same (Singleton confirmed).
```

Exercise 2: Implementing the Factory Method Pattern

```
package documentfactory;

public interface Document {

    void open();

    void save();

    void close();

    String getType();

}

public abstract class DocumentFactory {

    public abstract Document createDocument();

}

public class WordDocument implements Document {

    @Override

    public void open() {

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

    }
    @Override
    public void save() {
        System.out.println("Saving Word Document...");
    }
    @Override
    public void close() {
        System.out.println("Closing Word Document...");
    }
    @Override
    public String getType() {
        return "Word Document";
    }
}
public class WordDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new WordDocument();
    }
}
public class PdfDocument implements Document {
    @Override
    public void open() {
        System.out.println("Opening PDF Document...");
    }
    @Override
    public void save() {
        System.out.println("Saving PDF Document...");
    }
    @Override
    public void close() {
        System.out.println("Closing PDF Document...");
    }
    @Override
    public String getType() {
        return "PDF Document";
    }
}
public class PdfDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new PdfDocument();
    }
}
public class ExcelDocument implements Document {
    @Override
    public void open() {
        System.out.println("Opening Excel Document...");
    }
    @Override
    public void save() {
        System.out.println("Saving Excel Document...");
    }
    @Override
    public void close() {
        System.out.println("Closing Excel Document...");
    }
    @Override
    public String getType() {
        return "Excel Document";
    }
}
public class ExcelDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new ExcelDocument();
    }
}
package documentfactory;
public class Main {
    public static void main(String[] args) {
       //for worddocument
        DocumentFactory wordFactory = new WordDocumentFactory();
        Document wordDoc = wordFactory.createDocument();
        System.out.println("Created: " + wordDoc.getType());
        wordDoc.open();
        wordDoc.save();
        wordDoc.close();
        System.out.println();

        // for PDFdocument
        DocumentFactory pdfFactory = new PdfDocumentFactory();
        Document pdfDoc = pdfFactory.createDocument();
        System.out.println("Created: " + pdfDoc.getType());
        pdfDoc.open();
        pdfDoc.save();
        pdfDoc.close();
        System.out.println();
        //for exceldocument
        DocumentFactory excelFactory = new ExcelDocumentFactory();
        Document excelDoc = excelFactory.createDocument();
        System.out.println("Created: " + excelDoc.getType());
        excelDoc.open();
        excelDoc.save();
        excelDoc.close();
    }
}
```

OUTPUT:



```
        pdfDoc.close();
        System.out.println();

        //for exceldocument
        DocumentFactory excelFactory = new ExcelDocumentFactory();
        Document excelDoc = excelFactory.createDocument();
        System.out.println("Created: " + excelDoc.getType());
        excelDoc.open();
        excelDoc.save();
        excelDoc.close();
    }
}
```

```
<terminated> Main (1) [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe  (21-Jun-
Created: Word Document
Opening Word Document...
Saving Word Document...
Closing Word Document...

Created: PDF Document
Opening PDF Document...
Saving PDF Document...
Closing PDF Document...

Created: Excel Document
Opening Excel Document...
Saving Excel Document...
Closing Excel Document...
```

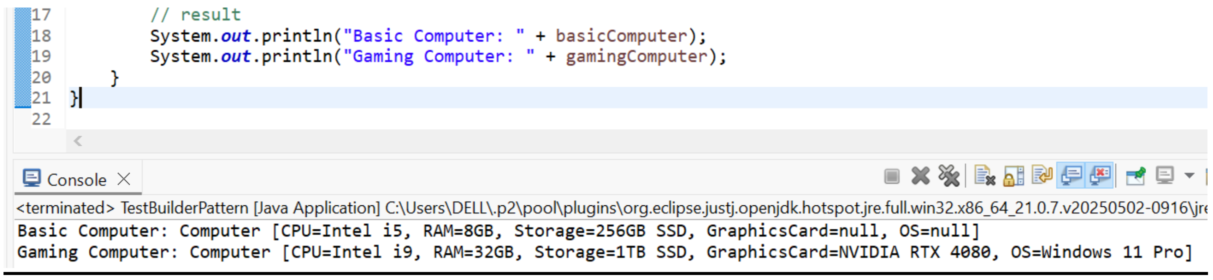
Exercise 3: Implementing the Builder Pattern

```
package builder;
public class Computer {
    // Required parameters
    private String CPU;
    private String RAM;
```

```
// Optional parameters
private String storage;
private String graphicsCard;
private String operatingSystem;

// Private constructor
private Computer(Builder builder) {
   this.CPU = builder.CPU;
   this.RAM = builder.RAM;
   this.storage = builder.storage;
   this.graphicsCard = builder.graphicsCard;
   this.operatingSystem = builder.operatingSystem;
}
// Static nested Builder class
public static class Builder {
   private String CPU;
   private String RAM;
   private String storage;
   private String graphicsCard;
   private String operatingSystem;
   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 setOperatingSystem(String operatingSystem) {
            this.operatingSystem = operatingSystem;
            return this;
        }
        public Computer build() {
            return new Computer(this);
        }
    }
    @Override
    public String toString() {
        return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", Storage=" + storage +",
GraphicsCard=" + graphicsCard + ", OS=" + operatingSystem + "]";
    }
}
public class TestBuilderPattern {
    public static void main(String[] args) {
        // Basic Computer Configuration
        Computer basicComputer = new Computer.Builder("Intel i5", "8GB")
            .setStorage("256GB SSD")
            .build();
        // Gaming Computer Configuration
        Computer gamingComputer = new Computer.Builder("Intel i9", "32GB")
            .setStorage("1TB SSD")
            .setGraphicsCard("NVIDIA RTX 4080")
            .setOperatingSystem("Windows 11 Pro")
            .build();
        // result
        System.out.println("Basic Computer: " + basicComputer);
        System.out.println("Gaming Computer: " + gamingComputer);
    }
}
```

OUTPUT:



```
        // result
        System.out.println("Basic Computer: " + basicComputer);
        System.out.println("Gaming Computer: " + gamingComputer);
    }
}
```

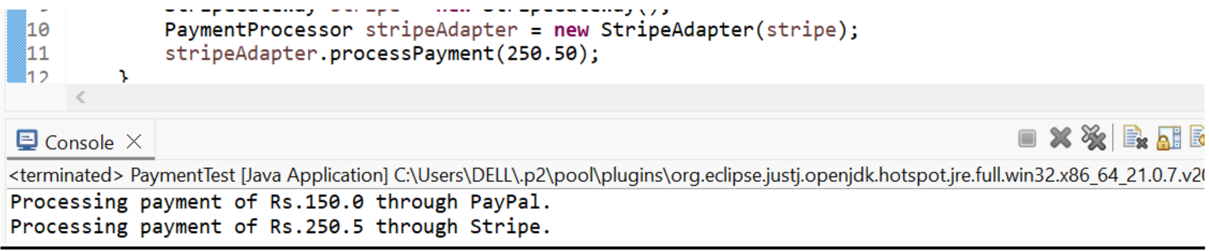
```
<terminated> TestBuilderPattern [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jr
Basic Computer: Computer [CPU=Intel i5, RAM=8GB, Storage=256GB SSD, GraphicsCard=null, OS=null]
Gaming Computer: Computer [CPU=Intel i9, RAM=32GB, Storage=1TB SSD, GraphicsCard=NVIDIA RTX 4080, OS=Windows 11 Pro]
```

Exercise 4: Implementing the Adapter Pattern

```
public interface PaymentProcessor {
    void processPayment(double amount);
}
public class PayPalAdapter implements PaymentProcessor {
    private PayPalGateway payPalGateway;
    public PayPalAdapter(PayPalGateway payPalGateway) {
        this.payPalGateway = payPalGateway;
    }
    @Override
    public void processPayment(double amount) {
        payPalGateway.sendPayment(amount);
    }
}
public class PayPalGateway {
    public void sendPayment(double amountInDollars) {
```

```java
        System.out.println("Processing payment of Rs." + amountInDollars + " through PayPal.");
    }
}
public class StripeAdapter implements PaymentProcessor {
    private StripeGateway stripeGateway;
    public StripeAdapter(StripeGateway stripeGateway) {
        this.stripeGateway = stripeGateway;
    }
    @Override
    public void processPayment(double amount) {
        stripeGateway.makePayment(amount);
    }
}
public class StripeGateway {
    public void makePayment(double money) {
        System.out.println("Processing payment of Rs." + money + " through Stripe.");
    }
}
public class PaymentTest {
    public static void main(String[] args) {
        // Using PayPal
        PayPalGateway payPal = new PayPalGateway();
        PaymentProcessor payPalAdapter = new PayPalAdapter(payPal);
        payPalAdapter.processPayment(150.00);

        // Using Stripe
        StripeGateway stripe = new StripeGateway();
        PaymentProcessor stripeAdapter = new StripeAdapter(stripe);
        stripeAdapter.processPayment(250.50);
    }
}
```

OUTPUT:



```
        PaymentProcessor stripeAdapter = new StripeAdapter(stripe);
        stripeAdapter.processPayment(250.50);
    }
```

Console

<terminated> PaymentTest [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v2

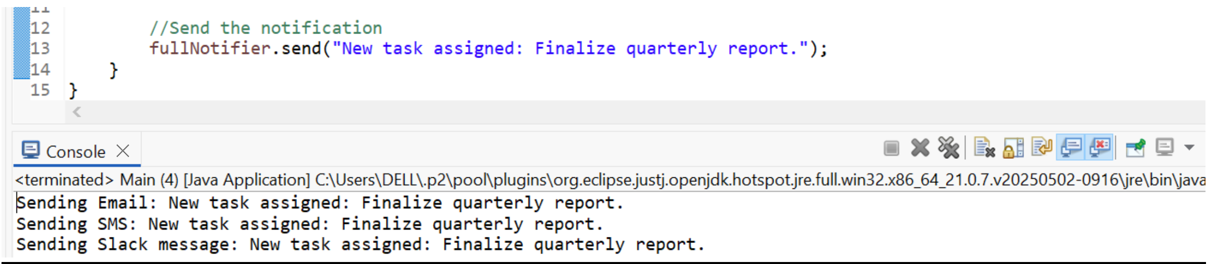
```
Processing payment of Rs.150.0 through PayPal.
Processing payment of Rs.250.5 through Stripe.
```

Exercise 5: Implementing the Decorator Pattern

```
public interface Notifier {
    void send(String message);
}
// EmailNotifier
public class EmailNotifier implements Notifier {
    @Override
    public void send(String message) {
        System.out.println("Sending Email: " + message);
    }
}
public abstract class NotifierDecorator implements Notifier {
    protected Notifier wrappedNotifier;
    public NotifierDecorator(Notifier notifier) {
        this.wrappedNotifier = notifier;
    }
    @Override
    public void send(String message) {
        wrappedNotifier.send(message);
    }
```

```
}
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);
    }
}
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);
    }
}

public class Main {
    public static void main(String[] args) {
        //Create the base notifier (Email)
        Notifier emailNotifier = new EmailNotifier();
        //Add SMS functionality
        Notifier smsAndEmailNotifier = new SMSNotifierDecorator(emailNotifier);

        // Add Slack functionality on top of Email + SMS
        Notifier fullNotifier = new SlackNotifierDecorator(smsAndEmailNotifier);

        //Send the notification
        fullNotifier.send("New task assigned: Finalize quarterly report.");
    }
}
```

OUTPUT:



```
        //Send the notification
        fullNotifier.send("New task assigned: Finalize quarterly report.");
    }
}
```

Console

```
<terminated> Main (4) [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\java
Sending Email: New task assigned: Finalize quarterly report.
Sending SMS: New task assigned: Finalize quarterly report.
Sending Slack message: New task assigned: Finalize quarterly report.
```

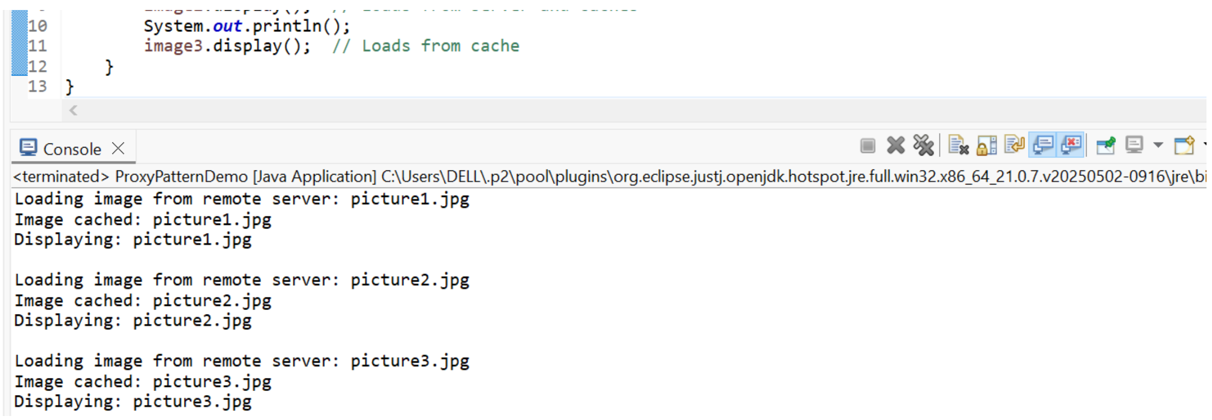
Exercise 6: Implementing the Proxy Pattern

```
import java.util.HashMap;
import java.util.Map;
public interface Image {
    void display();
}
public class ProxyImage implements Image {
    private String filename;
    private static Map<String, RealImage> cache = new HashMap<>();
```

```java
    public ProxyImage(String filename) {
        this.filename = filename;
    }
    @Override
    public void display() {
        RealImage realImage = cache.get(filename);
        if (realImage == null) {
            realImage = new RealImage(filename);
            cache.put(filename, realImage);
            System.out.println("Image cached: " + filename);
        } else {
            System.out.println("Image loaded from cache: " + filename);
        }
        realImage.display();
    }
}
public class RealImage implements Image {
    private String filename;
    public RealImage(String filename) {
        this.filename = filename;
        loadFromRemoteServer();
    }
    private void loadFromRemoteServer() {
        System.out.println("Loading image from remote server: " + filename);
    }
    @Override
    public void display() {
        System.out.println("Displaying: " + filename);
    }
```

```
}
public class ProxyPatternDemo {
    public static void main(String[] args) {
        Image image1 = new ProxyImage("picture1.jpg");
        Image image2 = new ProxyImage("picture2.jpg");
        Image image3 = new ProxyImage("picture3.jpg");
        image1.display();  // Loads from server and caches
        System.out.println();
        image2.display();  // Loads from server and caches
        System.out.println();
        image3.display();  // Loads from cache
    }
}
```

OUTPUT:



```
        System.out.println();
        image3.display();  // Loads from cache
    }
}
```

Console

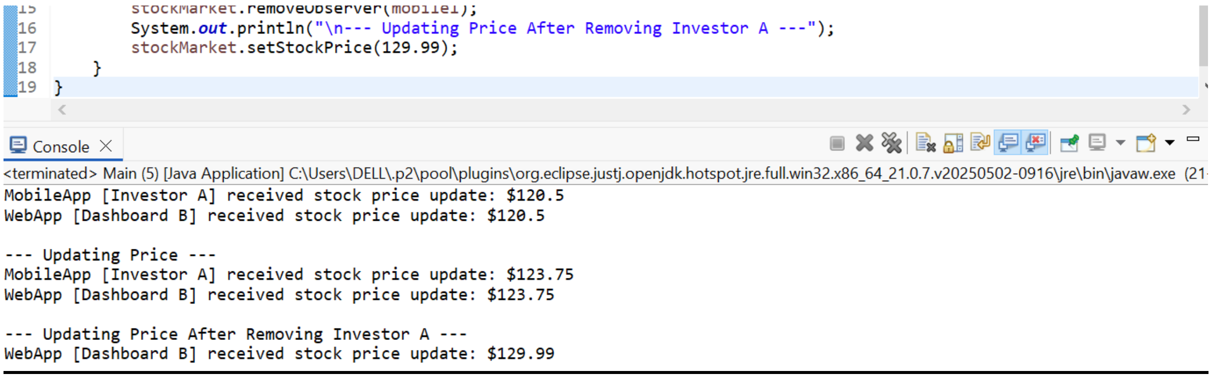
<terminated> ProxyPatternDemo [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bi

```
Loading image from remote server: picture1.jpg
Image cached: picture1.jpg
Displaying: picture1.jpg

Loading image from remote server: picture2.jpg
Image cached: picture2.jpg
Displaying: picture2.jpg

Loading image from remote server: picture3.jpg
Image cached: picture3.jpg
Displaying: picture3.jpg
```

Exercise 7: Implementing the Observer Pattern

```
import java.util.ArrayList;
import java.util.List;
public interface Stock {
    void registerObserver(Observer o);
    void removeObserver(Observer o);
    void notifyObservers();
}
public class StockMarket implements Stock {
    private List<Observer> observers = new ArrayList<>();
    private double stockPrice;
    @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(stockPrice);
        }
    }
    public void setStockPrice(double price) {
        this.stockPrice = price;
        notifyObservers();
    }
}
public interface Observer {
    void update(double price);
}
public class MobileApp implements Observer {
    private String name;
    public MobileApp(String name) {
        this.name = name;
    }
    @Override
    public void update(double price) {
        System.out.println("MobileApp [" + name + "] received stock price update: $" + price);
    }
}
public class WebApp implements Observer {
    private String name;
    public WebApp(String name) {
        this.name = name;
    }
    @Override
    public void update(double price) {
        System.out.println("WebApp [" + name + "] received stock price update: $" + price);
    }
}
public class Main {
    public static void main(String[] args) {
        StockMarket stockMarket = new StockMarket();
        Observer mobile1 = new MobileApp("Investor A");
        Observer web1 = new WebApp("Dashboard B");
        stockMarket.registerObserver(mobile1);
        stockMarket.registerObserver(web1);
        stockMarket.setStockPrice(120.50);
        System.out.println("\n--- Updating Price ---");
        stockMarket.setStockPrice(123.75);
        stockMarket.removeObserver(mobile1);
        System.out.println("\n--- Updating Price After Removing Investor A ---");
        stockMarket.setStockPrice(129.99);
    }
}
```

OUTPUT:



```
        stockMarket.removeObserver(mobile1);
        System.out.println("\n--- Updating Price After Removing Investor A ---");
        stockMarket.setStockPrice(129.99);
    }
}
```

Console

<terminated> Main (5) [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (21

```
MobileApp [Investor A] received stock price update: $120.5
WebApp [Dashboard B] received stock price update: $120.5

--- Updating Price ---
MobileApp [Investor A] received stock price update: $123.75
WebApp [Dashboard B] received stock price update: $123.75

--- Updating Price After Removing Investor A ---
WebApp [Dashboard B] received stock price update: $129.99
```

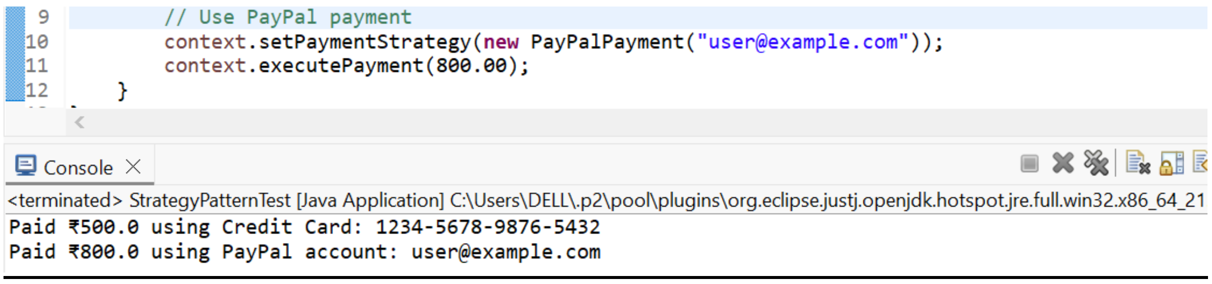
Exercise 8: Implementing the Strategy Pattern

```
public interface PaymentStrategy {
    void pay(double amount);
}
public class CreditCardPayment implements PaymentStrategy {
    private String cardNumber;
    private String cardHolderName;
    public CreditCardPayment(String cardNumber, String cardHolderName) {
        this.cardNumber = cardNumber;
        this.cardHolderName = cardHolderName;
    }
    @Override
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using Credit Card: " + cardNumber);
    }
}
```

```
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);
    }
}
public class PaymentContext {
    private PaymentStrategy paymentStrategy;
    public void setPaymentStrategy(PaymentStrategy paymentStrategy) {
        this.paymentStrategy = paymentStrategy;
    }
    public void executePayment(double amount) {
        if (paymentStrategy == null) {
            System.out.println("Payment strategy not set.");
        } else {
            paymentStrategy.pay(amount);
        }
    }
}
public class StrategyPatternTest {
    public static void main(String[] args) {
        PaymentContext context = new PaymentContext();

        // Use Credit Card payment
        context.setPaymentStrategy(new CreditCardPayment("1234-5678-9876-5432", "Vidya
CS"));
```

```
        context.executePayment(500.00);

        // Use PayPal payment
        context.setPaymentStrategy(new PayPalPayment("user@example.com"));
        context.executePayment(800.00);
    }
}
```

OUTPUT:



```
        // Use PayPal payment
        context.setPaymentStrategy(new PayPalPayment("user@example.com"));
        context.executePayment(800.00);
    }
```

<terminated> StrategyPatternTest [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21

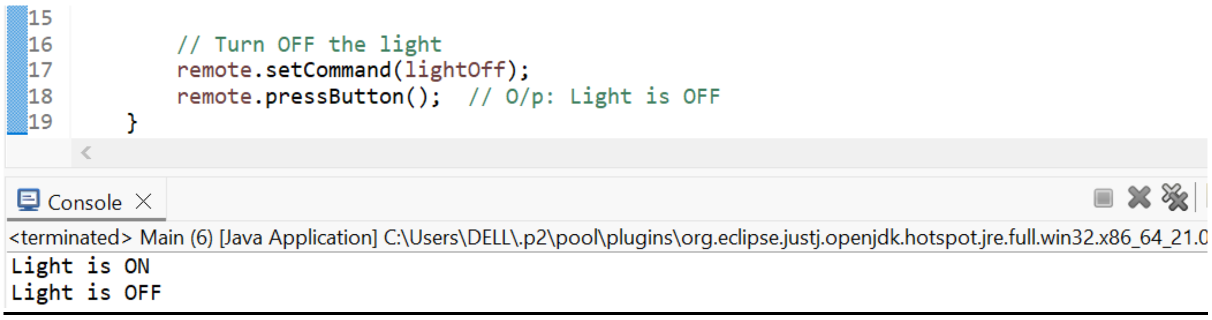
```
Paid ₹500.0 using Credit Card: 1234-5678-9876-5432
Paid ₹800.0 using PayPal account: user@example.com
```

Exercise 9: Implementing the Command Pattern

```
public interface Command {
    void execute();
}
public class LightOnCommand implements Command {
    private Light light;
    public LightOnCommand(Light light) {
        this.light = light;
    }
    public void execute() {
        light.turnOn();
    }
```

```
}
public class LightOffCommand implements Command {
   private Light light;
   public LightOffCommand(Light light) {
      this.light = light;
   }
   public void execute() {
      light.turnOff();
   }
}
public class RemoteControl {
   private Command command;
   public void setCommand(Command command) {
      this.command = command;
   }
   public void pressButton() {
      command.execute();
   }
}
public class Light {
   public void turnOn() {
      System.out.println("Light is ON");
   }
   public void turnOff() {
      System.out.println("Light is OFF");
   }
}
public class Main {
   public static void main(String[] args) {
      // Receiver
      Light livingRoomLight = new Light();

      // Concrete Commands
      Command lightOn = new LightOnCommand(livingRoomLight);
      Command lightOff = new LightOffCommand(livingRoomLight);
      RemoteControl remote = new RemoteControl();

      // Turn ON the light
      remote.setCommand(lightOn);
      remote.pressButton();  // O/p: Light is ON

      // Turn OFF the light
      remote.setCommand(lightOff);
      remote.pressButton();  // O/p: Light is OFF
   }
}
```

<u>OUTPUT:</u>



```

        // Turn OFF the light
        remote.setCommand(lightOff);
        remote.pressButton();  // O/p: Light is OFF
    }
```

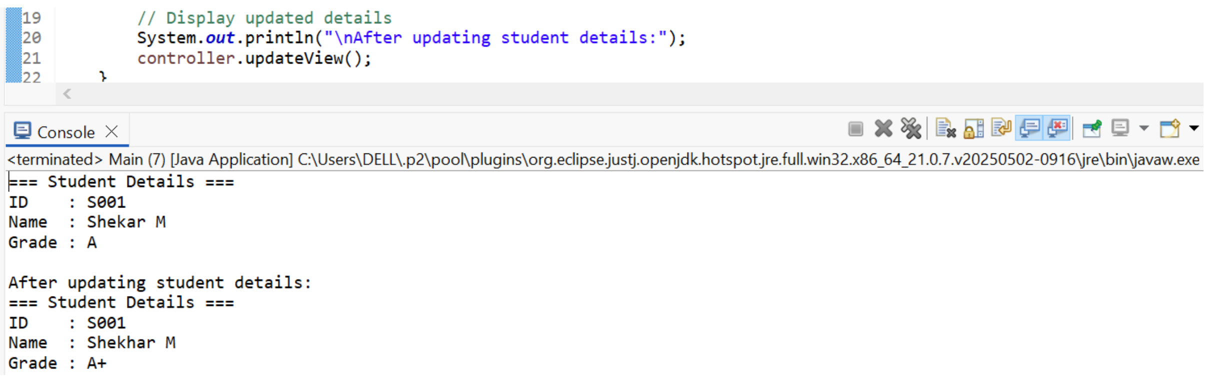
```
<terminated> Main (6) [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0
Light is ON
Light is OFF
```

Exercise 10: Implementing the MVC Pattern

```
public class Student {
```

```java
private String id;
private String name;
private String grade;

// Constructor
public Student(String id, String name, String grade) {
    this.id = id;
    this.name = name;
    this.grade = grade;
}

// Getters and Setters
public String getId() {
    return id;
}
public void setId(String id) {
    this.id = id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getGrade() {
    return grade;
}
public void setGrade(String grade) {
    this.grade = grade;
    }
}
public class StudentView {
    public void displayStudentDetails(String id, String name, String grade) {
        System.out.println("=== Student Details ===");
        System.out.println("ID    : " + id);
        System.out.println("Name  : " + name);
        System.out.println("Grade : " + grade);
    }
}
public class StudentController {
    private Student model;
    private StudentView view;
    public StudentController(Student model, StudentView view) {
        this.model = model;
        this.view = view;
    }
    // Update model data
    public void setStudentName(String name) {
        model.setName(name);
    }
    public void setStudentGrade(String grade) {
        model.setGrade(grade);
    }
    public void setStudentId(String id) {
        model.setId(id);
    }
    // Retrieve model data
    public String getStudentName() {
        return model.getName();
    }
    public String getStudentGrade() {
        return model.getGrade();
    }
    public String getStudentId() {
        return model.getId();
    }
    // Display updated data
    public void updateView() {
        view.displayStudentDetails(model.getId(), model.getName(), model.getGrade());
    }
}
public class Main {
    public static void main(String[] args) {
        // Create a student (Model)
        Student student = new Student("S001", "Shekar M", "A");

        // Create the view
        StudentView view = new StudentView();

        // Create the controller
        StudentController controller = new StudentController(student, view);

        // Initial display
        controller.updateView();

        // Update student details using controller
        controller.setStudentName("Shekhar M");
        controller.setStudentGrade("A+");

        // Display updated details
        System.out.println("\nAfter updating student details:");
        controller.updateView();
    }
}
```

OUTPUT:



```
        // Display updated details
        System.out.println("\nAfter updating student details:");
        controller.updateView();
    }
```

Console

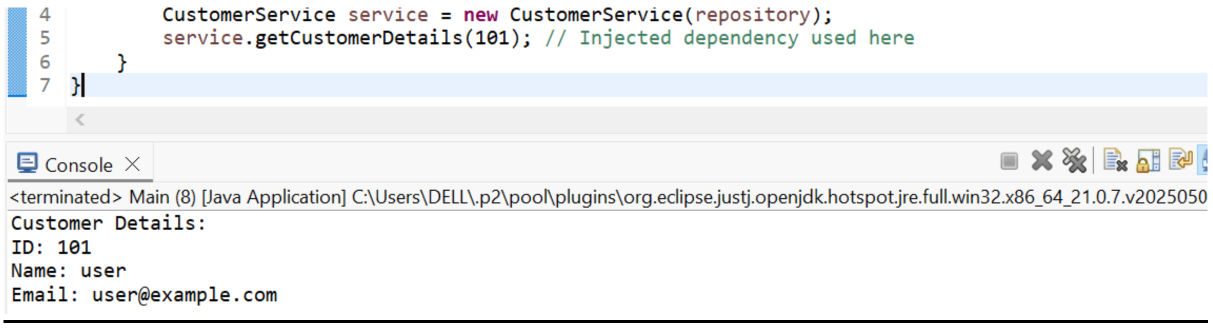
<terminated> Main (7) [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe

```
=== Student Details ===
ID    : S001
Name  : Shekar M
Grade : A

After updating student details:
=== Student Details ===
ID    : S001
Name  : Shekhar M
Grade : A+
```

Exercise 11: Implementing Dependency Injection

```
public interface CustomerRepository {
    Customer findCustomerById(int id);
}
public class CustomerRepositoryImpl implements CustomerRepository {
    @Override
    public Customer findCustomerById(int id) {
        return new Customer(id, "user", "user@example.com");
    }
}
public class CustomerService {
    private final 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 Details:");
        System.out.println(customer);
    }
}
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;
    }
    @Override
    public String toString() {
        return "ID: " + id + "\nName: " + name + "\nEmail: " + email;
    }
}
public class Main {
    public static void main(String[] args) {
        CustomerRepository repository = new CustomerRepositoryImpl();
        CustomerService service = new CustomerService(repository);
        service.getCustomerDetails(101); // Injected dependency used here
    }
}
```

OUTPUT:



```
        CustomerService service = new CustomerService(repository);
        service.getCustomerDetails(101); // Injected dependency used here
    }
}
```

```
<terminated> Main (8) [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v2025050
Customer Details:
ID: 101
Name: user
Email: user@example.com
```