

### SKILL AOOP WEEK-3

1. Adapting Different Payment Gateways Objective: Integrate multiple payment gateways (e.g., PayPal, Stripe) into a single payment processing interface. a. Create Interfaces: Define a PaymentProcessor interface with methods like processPayment(amount: double). b. Implement Concrete Adapters: Implement adapters for each payment gateway (e.g., PayPalAdapter, StripeAdapter) that adapt their specific APIs to the PaymentProcessor interface. c. Test Integration: Create a client class that uses the PaymentProcessor interface to process payments, and test with different adapters. d. Analyze the issue in this scenario that the Adapter pattern is meant to address.

**Code:**

```
package week3;
```

```
public class StripeAdapter implements PaymentProcessor {
    private StripeAPI stripeAPI;

    public StripeAdapter(StripeAPI stripeAPI) {
        this.stripeAPI = stripeAPI;
    }

    @Override
    public void processPayment(double amount) {
        stripeAPI.charge(amount);
    }
}
```

```
package week3;
```

```
public class StripeAPI {
    public void charge(double amount) {
        System.out.println("Processing payment of Rs" + amount + " through Stripe.");
    }
}
```

```
package week3;
```

```
public class PayPalAPI {
    public void makePayment(double amount) {
        System.out.println("Processing payment of Rs" + amount + " through PayPal.");
    }
}
```

```
package week3;
```

```
public class PayPalAdapter implements PaymentProcessor {
    private PayPalAPI payPalAPI;
```

```
    public PayPalAdapter(PayPalAPI payPalAPI) {
        this.payPalAPI = payPalAPI;
```

```

    }

    @Override
    public void processPayment(double amount) {
        payPalAPI.makePayment(amount);
    }
}

```

**package** week3;

```

public interface PaymentProcessor {
    void processPayment(double amount);
}
package week3;

```

```

public class PaymentClient {
    private PaymentProcessor paymentProcessor;

    public PaymentClient(PaymentProcessor paymentProcessor) {
        this.paymentProcessor = paymentProcessor;
    }

    public void makePayment(double amount) {
        paymentProcessor.processPayment(amount);
    }

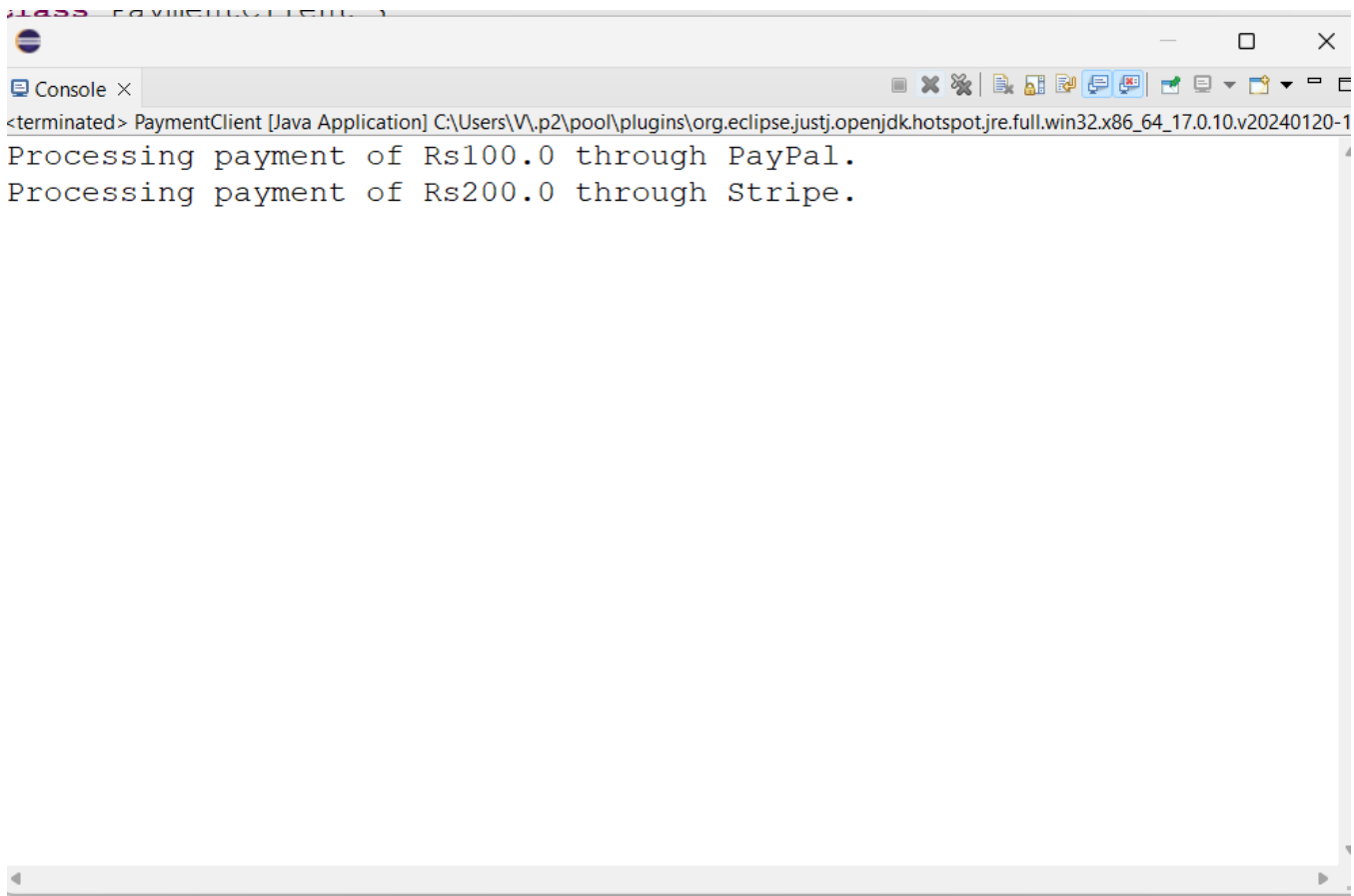
    public static void main(String[] args) {

        PayPalAPI payPalAPI = new PayPalAPI();
        PaymentProcessor payPalAdapter = new PayPalAdapter(payPalAPI);
        PaymentClient payPalClient = new PaymentClient(payPalAdapter);
        payPalClient.makePayment(100.00);

        StripeAPI stripeAPI = new StripeAPI();
        PaymentProcessor stripeAdapter = new StripeAdapter(stripeAPI);
        PaymentClient stripeClient = new PaymentClient(stripeAdapter);
        stripeClient.makePayment(200.00);
    }
}

```

Output:



2. Extending Functionality of Text Processing Objective: Enhance a basic text processing class with additional features (e.g., spell checking, text formatting).
  - a. Create Base Component: Define a TextProcessor class with a method process (text: String).
  - b. Create Decorators: Implement decorators like SpellCheckDecorator, TextFormatDecorator that extend TextProcessor and add new functionalities.
  - c. Test Decorators: Create a client class that uses decorated TextProcessor objects to process text with added functionalities.
  - d. Analyze the issue in this scenario that the Decorator pattern is meant to address.

CODE:

```
package week3;
```

```
public class SpellCheckDecorator extends TextProcessorDecorator {
```

```
    public SpellCheckDecorator(TextProcessor textProcessor) {  
        super(textProcessor);  
    }
```

```
    @Override  
    public String process(String text) {  
        text = super.process(text);  
        return spellCheck(text);  
    }
```

```
    private String spellCheck(String text) {  
        return text.replaceAll("teh", "the");  
    }  
}
```

```
package week3;
```

```
public class TextFormatDecorator extends TextProcessorDecorator {
```

```
    public TextFormatDecorator(TextProcessor textProcessor) {  
        super(textProcessor);  
    }
```

```
    @Override  
    public String process(String text) {  
        text = super.process(text);  
        return formatText(text);  
    }
```

```
    private String formatText(String text) {  
  
        return text.toUpperCase();  
    }  
}
```

```
package week3;
```

```
public class TextProcessor {  
    public String process(String text) {  
        return text;  
    }  
}
```

```
package week3;
```

```
public abstract class TextProcessorDecorator extends TextProcessor {  
    protected TextProcessor textProcessor;
```

```
    public TextProcessorDecorator(TextProcessor textProcessor) {  
        this.textProcessor = textProcessor;  
    }
```

```
    @Override  
    public String process(String text) {  
        return textProcessor.process(text);  
    }  
}
```

```
package week3;
```

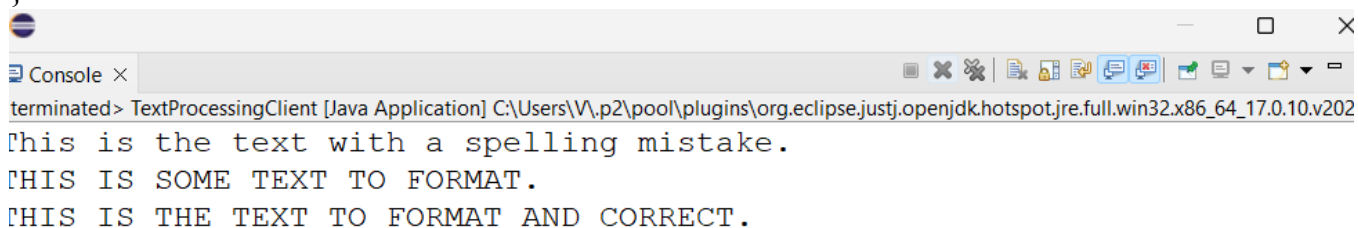
```
public class TextProcessingClient {  
    public static void main(String[] args) {  
        TextProcessor basicProcessor = new TextProcessor();
```

```
        TextProcessor spellCheckedProcessor = new SpellCheckDecorator(basicProcessor);
```

```
System.out.println(spellCheckedProcessor.process("This is teh text with a spelling mistake."));
```

```
TextProcessor formattedProcessor = new TextFormatDecorator(basicProcessor);  
System.out.println(formattedProcessor.process("This is some text to format."));
```

```
TextProcessor fullyDecoratedProcessor = new TextFormatDecorator(new  
SpellCheckDecorator(basicProcessor));  
System.out.println(fullyDecoratedProcessor.process("This is teh text to format and  
correct."));  
}  
}
```



The screenshot shows the Eclipse IDE's console window. The title bar indicates the application is 'TextProcessingClient [Java Application]' running on 'C:\Users\V\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_17.0.10.v202...'. The console output displays three lines of text: 'This is the text with a spelling mistake.', 'THIS IS SOME TEXT TO FORMAT.', and 'THIS IS THE TEXT TO FORMAT AND CORRECT.'. The text is formatted with color: the first line is in blue, the second in red, and the third in green.

```
terminated> TextProcessingClient [Java Application] C:\Users\V\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v202  
This is the text with a spelling mistake.  
THIS IS SOME TEXT TO FORMAT.  
THIS IS THE TEXT TO FORMAT AND CORRECT.
```

3. Template Design Pattern Experiment Objective: Implement a simple framework for data processing tasks using the Template Method pattern. a. Create an abstract class `DataProcessor` with a template method `process()` that defines the steps for processing data (e.g., `loadData`, `processData`, `saveData`). b. Implement at least two concrete subclasses (`CSVDataProcessor` and `JSONDataProcessor`) that override the necessary steps, providing specific implementations for handling CSV and JSON data formats. c. Analyze the issue in this scenario that the Template pattern is meant to address.

CODE:

```
package week3;
```

```
abstract class DataProcessor {
```

```

public final void process() {
    loadData();
    processData();
    saveData();
}

protected abstract void loadData();

protected abstract void processData();

protected abstract void saveData();
}
package week3;

class JSONDataProcessor extends DataProcessor {

    @Override
    protected void loadData() {
        System.out.println("Loading JSON data...");
    }

    @Override
    protected void processData() {
        System.out.println("Processing JSON data...");
    }

    @Override
    protected void saveData() {
        System.out.println("Saving processed JSON data...");
    }
}
package week3;

class CSVDataProcessor extends DataProcessor {

    @Override
    protected void loadData() {
        System.out.println("Loading CSV data...");
    }

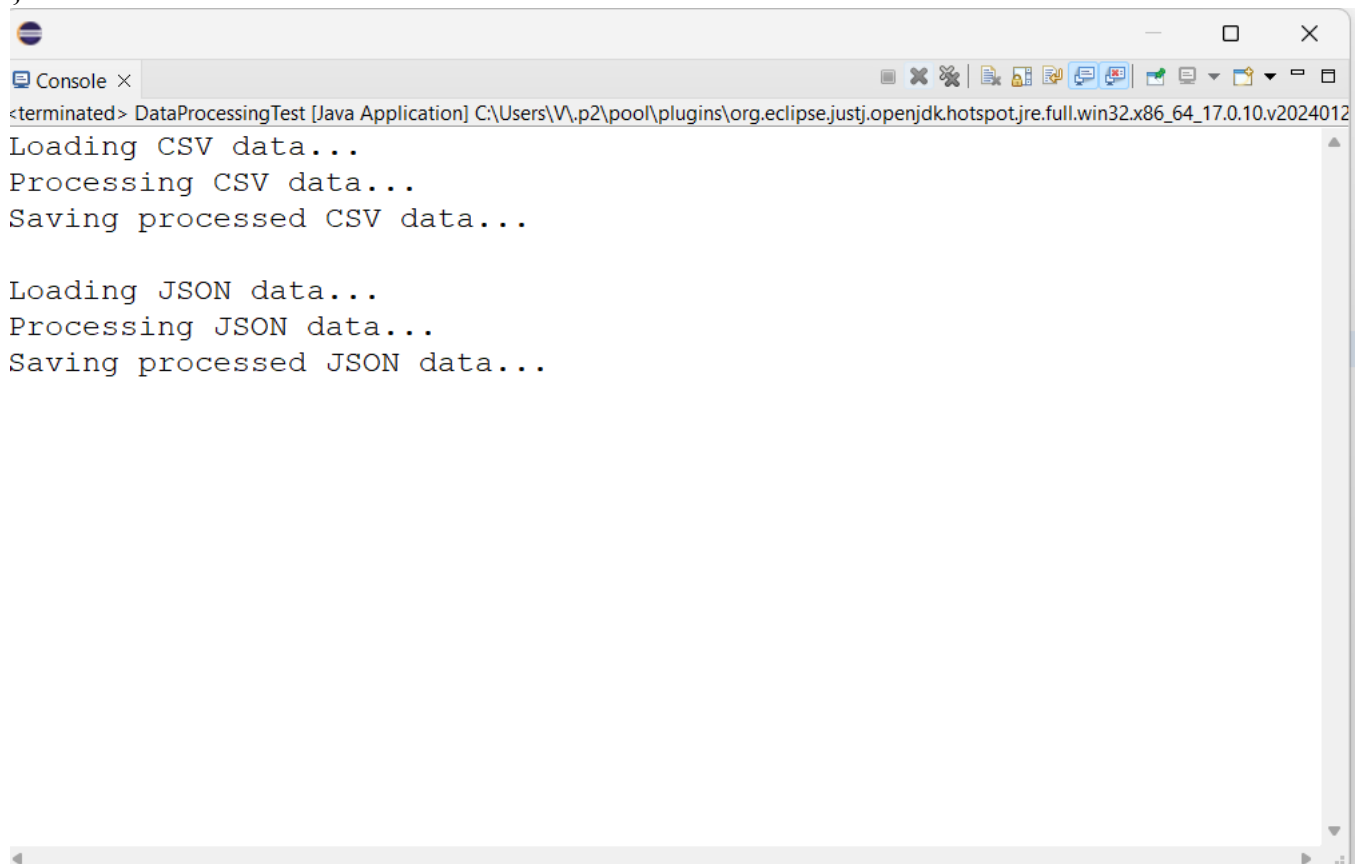
    @Override
    protected void processData() {
        System.out.println("Processing CSV data...");
    }

    @Override
    protected void saveData() {
        System.out.println("Saving processed CSV data...");
    }
}

```

```
package week3;
```

```
public class DataProcessingTest {  
    public static void main(String[] args) {  
        DataProcessor csvProcessor = new CSVDataProcessor();  
        csvProcessor.process();  
  
        System.out.println();  
  
        DataProcessor jsonProcessor = new JSONDataProcessor();  
        jsonProcessor.process();  
    }  
}
```



```
<terminated> DataProcessingTest [Java Application] C:\Users\V\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v2024012  
Loading CSV data...  
Processing CSV data...  
Saving processed CSV data...  
  
Loading JSON data...  
Processing JSON data...  
Saving processed JSON data...
```

4. Request Processing Pipeline Objective: Implement a request processing pipeline where each handler processes a request or passes it to the next handler. a. Create Handler Interface: Define a Handler interface with a method `handleRequest(request: Request)`. b. Implement Concrete Handlers: Implement handlers like `AuthenticationHandler`, `ValidationHandler`, `BusinessLogicHandler` that process specific aspects of the request. c. Create Chain: Link handlers together to form a chain where each handler can pass the request to the next handler if it cannot handle it. d. Test Chain: Create a client class that sends requests through the chain and observes how they are processed by different handlers. e. Analyze the issue in this scenario that the Chain of Responsibility pattern is meant to address.

CODE:

```
package week3;
```

```
public interface Handler {  
    void handleRequest(Request request);
```

```

        Handler getNextHandler();

        void setNextHandler(Handler handler);
    }
package week3;

```

```

public class AuthenticationHandler implements Handler {
    private Handler nextHandler;

```

```

    public void setNextHandler(Handler nextHandler) {
        this.nextHandler = nextHandler;
    }

```

```

    @Override
    public void handleRequest(Request request) {
        if (!request.isAuthenticated()) {
            System.out.println("Authenticating request: " + request.getDescription());
            request.setAuthenticated(true);
        }
        if (nextHandler != null) {
            nextHandler.handleRequest(request);
        }
    }

```

```

    @Override
    public Handler getNextHandler() {
        // TODO Auto-generated method stub
        return null;
    }
}

```

```

package week3;

```

```

public class BusinessLogicHandler implements Handler {
    private Handler nextHandler;

```

```

    public void setNextHandler(Handler nextHandler) {
        this.nextHandler = nextHandler;
    }

```

```

    @Override
    public void handleRequest(Request request) {
        if (!request.isProcessed()) {
            System.out.println("Processing business logic for request: " + request.getDescription());
            request.setProcessed(true);
        }
        if (nextHandler != null) {
            nextHandler.handleRequest(request);
        }
    }

```



```

    }
}

@Override
public Handler getNextHandler() {
    // TODO Auto-generated method stub
    return null;
}
}

package week3;

public class ValidationHandler implements Handler {
    private Handler nextHandler;

    public void setNextHandler(Handler nextHandler) {
        this.nextHandler = nextHandler;
    }

    @Override
    public void handleRequest(Request request) {
        if (!request.isValidated()) {
            System.out.println("Validating request: " + request.getDescription());
            request.setValidated(true);
        }
        if (nextHandler != null) {
            nextHandler.handleRequest(request);
        }
    }

    @Override
    public Handler getNextHandler() {
        // TODO Auto-generated method stub
        return null;
    }
}

package week3;

public class HandlerChain {
    private Handler firstHandler;

    public void addHandler(Handler handler) {
        if (firstHandler == null) {
            firstHandler = handler;
        } else {
            Handler current = firstHandler;
            while (current instanceof Handler) {
                Handler nextHandler = ((Handler) current).getNextHandler();
                if (nextHandler == null) {
                    ((Handler) current).setNextHandler(handler);
                }
            }
        }
    }
}

```

```

        break;
    }
    current = nextHandler;
}
}
}

public void handleRequest(Request request) {
    if (firstHandler != null) {
        firstHandler.handleRequest(request);
    }
}
}

package week3;

public class Request {
    private String description;
    private boolean authenticated;
    private boolean validated;
    private boolean processed;

    public Request(String description) {
        this.description = description;
    }

    public String getDescription() {
        return description;
    }

    public boolean isAuthenticated() {
        return authenticated;
    }

    public void setAuthenticated(boolean authenticated) {
        this.authenticated = authenticated;
    }

    public boolean isValidated() {
        return validated;
    }

    public void setValidated(boolean validated) {
        this.validated = validated;
    }

    public boolean isProcessed() {
        return processed;
    }
}

```

```

    public void setProcessed(boolean processed) {
        this.processed = processed;
    }
}
package week3;

public class RequestProcessingClient {
    public static void main(String[] args) {

        AuthenticationHandler authHandler = new AuthenticationHandler();
        ValidationHandler validationHandler = new ValidationHandler();
        BusinessLogicHandler businessLogicHandler = new BusinessLogicHandler();

        authHandler.setNextHandler(validationHandler);
        validationHandler.setNextHandler(businessLogicHandler);

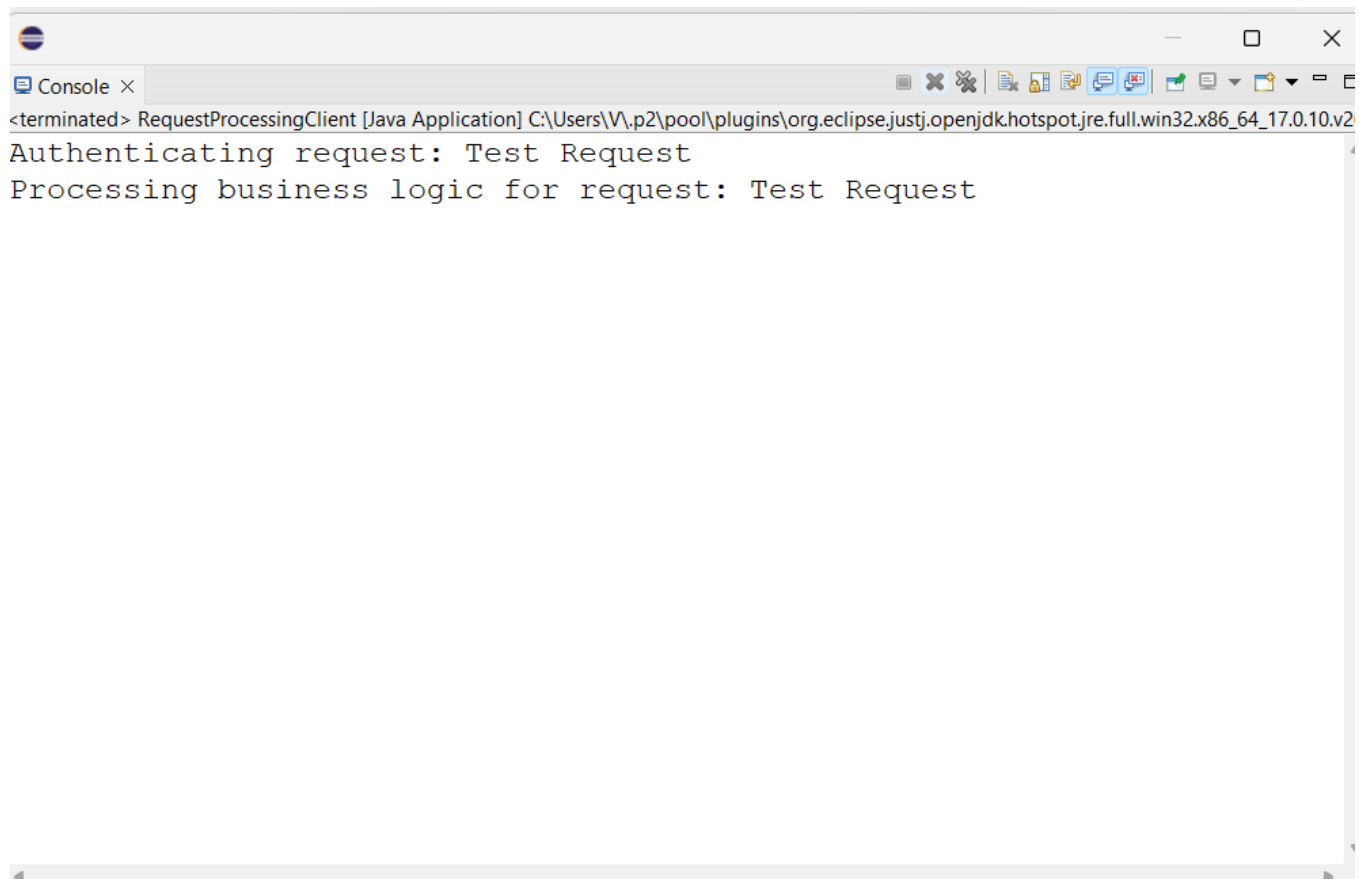
        HandlerChain handlerChain = new HandlerChain();
        handlerChain.addHandler(authHandler);
        handlerChain.addHandler(validationHandler);
        handlerChain.addHandler(businessLogicHandler);

        Request request = new Request("Test Request");

        handlerChain.handleRequest(request);
    }
}

```

OUTPUT:



```
<terminated> RequestProcessingClient [Java Application] C:\Users\V\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v2
Authenticating request: Test Request
Processing business logic for request: Test Request
```