Scenario 8: Customer Request

Customer Request:

"I need a document editor that allows users to process different types of documents, such as Word, PDF, and Text, through various steps like parsing, validating, and formatting. However, I want the processing steps to be customizable and able to vary independently. For example, a user might want to parse a document without validating it, or apply formatting after validation."

Choosing the Right Pattern:

Pattern Recommendation: Chain of Responsibility Pattern

Why?

The Chain of Responsibility Pattern is ideal for this scenario because:

- Flexible Processing Steps: The customer needs a way to process documents through different steps that can be combined, skipped, or reordered as needed. The Chain of Responsibility Pattern allows each processing step to be handled by a separate handler, which can pass the document along the chain to the next handler or stop the processing.

- Customization: The customer wants the ability to customize the processing pipeline. The Chain of Responsibility Pattern allows handlers to be chained together in various configurations, making it easy to customize the processing steps without altering the core logic.

- Decoupling: The pattern decouples the sender of a request from its receivers by allowing multiple objects to handle the request. This makes the system more modular and easier to extend.

Pattern Explanation: Chain of Responsibility Pattern

Key Concepts:

- Handler: An interface that defines a method for handling requests and a reference to the next handler in the chain.
- Concrete Handler: Implements the handler interface, processes the request, and optionally passes it to the next handler in the chain.
- Client: Sends the request to the first handler in the chain. The request is then passed along the chain until a handler processes it or the end of the chain is reached.

How It Works:

- Handler Interface: Defines methods like handleRequest() that are implemented by concrete handlers to process the request.
- Concrete Handler: Implements the Handler interface and either handles the request or passes it to the next handler in the chain.
- Client: Configures the chain by linking handlers together and sends the request to the first handler in the chain.

Implementation Example:

```
// Handler Interface
interface DocumentProcessor {
    void setNext(DocumentProcessor nextProcessor);
    void process(Document document);
}
```

```
// Concrete Handlers
class ParsingProcessor implements DocumentProcessor {
    private DocumentProcessor nextProcessor;

    @Override
    public void setNext(DocumentProcessor nextProcessor) {
        this.nextProcessor = nextProcessor;
    }

    @Override
    public void process(Document document) {
        System.out.println("Parsing document: " + document.getName());
        if (nextProcessor != null) {
            nextProcessor.process(document);
        }
    }
}

class ValidationProcessor implements DocumentProcessor {
    private DocumentProcessor nextProcessor;

    @Override
    public void setNext(DocumentProcessor nextProcessor) {
        this.nextProcessor = nextProcessor;
    }
```

```
    @Override
    public void process(Document document) {
        System.out.println("Validating document: " + document.getName());
        if (nextProcessor != null) {
            nextProcessor.process(document);
        }
    }
}

class FormattingProcessor implements DocumentProcessor {
    private DocumentProcessor nextProcessor;

    @Override
    public void setNext(DocumentProcessor nextProcessor) {
        this.nextProcessor = nextProcessor;
    }

    @Override
    public void process(Document document) {
        System.out.println("Formatting document: " + document.getName());
        if (nextProcessor != null) {
            nextProcessor.process(document);
        }
    }
}
```

```
// Document Class
class Document {
    private String name;

    public Document(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}

// Client Code
public class DocumentEditor {
    public static void main(String[] args) {
        Document document = new Document("DesignPatterns.docx");

        // Set up the chain of responsibility
        DocumentProcessor parsing = new ParsingProcessor();
        DocumentProcessor validation = new ValidationProcessor();
        DocumentProcessor formatting = new FormattingProcessor();

        parsing.setNext(validation);
        validation.setNext(formatting);
```

```
        // Start the chain
        parsing.process(document);
    }
}
```