Scenario 15: Customer Request

Customer Request:

"I need a document editor that allows users to build documents from a series of components like headers, f

Choosing the Right Pattern:

Pattern Recommendation: Builder Pattern

Why?

The Builder Pattern is ideal for this scenario because:

Step-by-Step Construction: The customer wants to create complex objects (documents) step by step, with

Separation of Concerns: The pattern separates the construction of a complex object from its representation

Improved Readability and Maintenance: The Builder Pattern simplifies the creation of complex objects, mal

Pattern Explanation: Builder Pattern

Key Concepts:

Builder: An interface that defines the steps required to build the product (e.g., buildHeader(), buildFooter(),

Concrete Builder: Implements the Builder interface and provides specific implementations for building the c

Director: An object that constructs the product using the Builder interface. It directs the building process by

Product: The final object that is being built (e.g., a Document).

How It Works:

Builder Interface: Defines the methods needed to build the product (e.g., buildHeader(), buildFooter()).

Concrete Builder: Implements the Builder interface and provides the specific steps to construct the product

Director: The director uses the builder to construct the product by calling the builder's methods in a particul

Product: The complex object that is constructed by the builder (e.g., Document).

Client: The client creates the builder and director, and then initiates the construction process.

Implementation Example

Here's how the implementation might look in Java:

```
// Product
class Document {
    private String header;
    private String footer;
    private String content;

    public void setHeader(String header) {
        this.header = header;
    }

    public void setFooter(String footer) {
        this.footer = footer;
    }

    public void setContent(String content) {
```

```
        this.content = content;
    }

    @Override
    public String toString() {
        return header + "\n" + content + "\n" + footer;
    }
}

// Builder Interface
interface DocumentBuilder {
    void buildHeader(String header);
    void buildFooter(String footer);
    void buildContent(String content);
    Document getDocument();
}

// Concrete Builder
class SimpleDocumentBuilder implements DocumentBuilder {
    private Document document;

    public SimpleDocumentBuilder() {
        this.document = new Document();
    }

    @Override
```

```
    public void buildHeader(String header) {
        document.setHeader(header);
    }

    @Override
    public void buildFooter(String footer) {
        document.setFooter(footer);
    }

    @Override
    public void buildContent(String content) {
        document.setContent(content);
    }

    @Override
    public Document getDocument() {
        return document;
    }
}

// Director
class DocumentDirector {
    private DocumentBuilder builder;

    public DocumentDirector(DocumentBuilder builder) {
        this.builder = builder;
```

```
    }

    public void constructDocument(String header, String content, String footer) {
        builder.buildHeader(header);
        builder.buildContent(content);
        builder.buildFooter(footer);
    }
}

// Client Code
public class DocumentEditor {
    public static void main(String[] args) {
        DocumentBuilder builder = new SimpleDocumentBuilder();
        DocumentDirector director = new DocumentDirector(builder);

        director.constructDocument("Title: Design Patterns", "This is a sample content.", "Footer: Confidential

        Document document = builder.getDocument();
        System.out.println(document);
    }
}
```

Key Points to Remember:

Builder Pattern allows you to construct complex objects step by step. The construction process can be cus

It is particularly useful when the construction of an object involves several steps and when the object needs

When to Use the Builder Pattern:

When you need to construct complex objects step by step.

When you want to create different representations of the same object using the same construction process

When the construction process involves multiple steps and you want to separate this process from the fina

Advantages:

Step-by-Step Construction: The pattern allows you to construct objects gradually, making it easy to manag

Separation of Construction and Representation: The builder pattern separates the construction of an objec

Improves Readability: The pattern makes the code more readable and maintainable by breaking down the

Disadvantages:

Increased Complexity: The pattern can introduce additional complexity by requiring multiple builder classes

Overhead for Simple Objects: If the object being constructed is simple, using the Builder Pattern may introd