Template Method Pattern

Scenario:

Imagine you're developing a document editor that supports exporting documents to multiple formats like PDF, Word, and HTML. Each format has a slightly different export process, but the overall steps (such as preparing the document, converting the content, and saving the file) are the same. You want to enforce a consistent export process while allowing the specific steps to be customized for each format.

Purpose:

The Template Method Pattern defines the skeleton of an algorithm in a superclass, allowing subclasses to override specific steps of the algorithm without changing its overall structure. This pattern promotes code reuse and enforces a consistent workflow, while still allowing flexibility for subclasses to provide custom behavior.

When to Use:

When you have multiple classes that share the same algorithm but require specific implementations for certain steps.

When you want to enforce a consistent sequence of steps in an algorithm, while allowing subclasses to customize individual steps.

When you want to reduce code duplication by moving common code to a superclass and letting subclasses handle the details.

Key Concepts:

Template Method: A method in the superclass that defines the overall structure of the algorithm. It calls other methods (some of which may be abstract or hook methods) that can be implemented or

overridden by subclasses.

Concrete Methods: Methods in the superclass that provide a default implementation and are called by the template method.

Abstract Methods: Methods declared in the superclass that must be implemented by subclasses. These represent steps in the algorithm that are customized by subclasses.

Hook Methods: Optional methods that can be overridden by subclasses, providing flexibility in the algorithm without being mandatory.

How It Works:

Abstract Class: Contains the template method, which defines the skeleton of the algorithm. It may also include abstract methods for steps that subclasses must implement and concrete methods for common steps.

Concrete Subclasses: Implement or override the abstract methods and hook methods to provide specific behavior for each variation of the algorithm.

Client: Calls the template method on an instance of a concrete subclass, executing the algorithm as defined in the superclass with the customized behavior provided by the subclass.

Implementation Example:

Here's how the implementation might look in Java:

```
// Abstract Class
abstract class DocumentExporter {
    // Template Method
    public final void exportDocument() {
```

```
        prepareDocument();
        convertContent();
        saveFile();
        finalizeExport();
    }

    // Concrete Method
    protected void prepareDocument() {
        System.out.println("Preparing the document...");
    }

    // Abstract Methods (Steps that must be implemented by subclasses)
    protected abstract void convertContent();
    protected abstract void saveFile();

    // Hook Method (Optional step that can be overridden)
    protected void finalizeExport() {
        System.out.println("Finalizing export process...");
    }
}

// Concrete Subclass for PDF Export
class PDFExporter extends DocumentExporter {
    @Override
    protected void convertContent() {
        System.out.println("Converting content to PDF format...");
```

```
    }

    @Override
    protected void saveFile() {
        System.out.println("Saving PDF file...");
    }

    @Override
    protected void finalizeExport() {
        System.out.println("Performing additional PDF-specific finalization...");
    }
}

// Concrete Subclass for Word Export
class WordExporter extends DocumentExporter {
    @Override
    protected void convertContent() {
        System.out.println("Converting content to Word format...");
    }

    @Override
    protected void saveFile() {
        System.out.println("Saving Word file...");
    }
}
```

```
// Concrete Subclass for HTML Export
class HTMLExporter extends DocumentExporter {
    @Override
    protected void convertContent() {
        System.out.println("Converting content to HTML format...");
    }

    @Override
    protected void saveFile() {
        System.out.println("Saving HTML file...");
    }
}

// Client Code
public class DocumentEditor {
    public static void main(String[] args) {
        DocumentExporter pdfExporter = new PDFExporter();
        DocumentExporter wordExporter = new WordExporter();
        DocumentExporter htmlExporter = new HTMLExporter();

        System.out.println("Exporting to PDF:");
        pdfExporter.exportDocument();

        System.out.println("\nExporting to Word:");
        wordExporter.exportDocument();
```

```
        System.out.println("\nExporting to HTML:");
        htmlExporter.exportDocument();
    }
}
```

Key Points to Remember:

Template Method Pattern defines the structure of an algorithm in a method, allowing subclasses to override specific steps without altering the algorithm's overall structure. This pattern ensures consistency across different implementations while promoting code reuse.

Hook Methods: These are optional steps that subclasses can override to provide additional behavior. They offer flexibility without requiring all subclasses to implement them.

Advantages:

- Code Reuse: The pattern allows you to move common code to a superclass, reducing duplication across subclasses.
- Consistency: Ensures that the overall algorithm remains consistent across different implementations.
- Flexibility: Subclasses can customize specific steps of the algorithm without changing its structure.

Disadvantages:

- Limited Flexibility: The pattern can make the design less flexible if the algorithm needs to be completely changed, as it is tightly coupled to the superclass's structure.
- Dependency on Superclass: Subclasses are dependent on the structure defined by the superclass, which can lead to challenges if the superclass needs to change.