Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITC802 **DATE:** 18-03-2024

COURSE NAME: Design Patterns Laboratory **CLASS:** BE - IT

EXPERIMENT NO. 6

CO/LO: Identify and apply the most suitable design pattern to address a given application design problem.

AIM: Implement the Template Pattern using any language of your choice for any real-life scenario.

DESCRIPTION:

The Template Method design pattern is a behavioral design pattern that defines the skeleton of an algorithm in a superclass but allows subclasses to override specific steps of the algorithm without changing its structure. It promotes code reuse by encapsulating the common algorithmic structure in the superclass while allowing subclasses to provide concrete implementations for certain steps, thus enabling customization and flexibility.

SOURCE CODE:

```
from abc import ABC, abstractmethod

# Abstract class defining the template for document processing
class DocumentProcessor(ABC):
    def process_document(self, document):
        self.load_document(document)
        self.extract_content()
        self.analyze_content()
        self.save_results()

    @abstractmethod
    def load_document(self, document):
        pass
```

```
    @abstractmethod
    def extract_content(self):
        pass

    @abstractmethod
    def analyze_content(self):
        pass

    @abstractmethod
    def save_results(self):
        pass

# Concrete subclass for processing reports
class ReportProcessor(DocumentProcessor):
    def load_document(self, document):
        print("Loading report document...")

    def extract_content(self):
        print("Extracting content from report...")

    def analyze_content(self):
        print("Analyzing report content...")

    def save_results(self):
        print("Saving report analysis results...")

# Concrete subclass for processing invoices
class InvoiceProcessor(DocumentProcessor):
    def load_document(self, document):
        print("Loading invoice document...")

    def extract_content(self):
```

```
        print("Extracting content from invoice...")

    def analyze_content(self):
        print("Analyzing invoice content...")

    def save_results(self):
        print("Saving invoice analysis results...")

# Concrete subclass for processing contracts
class ContractProcessor(DocumentProcessor):
    def load_document(self, document):
        print("Loading contract document...")

    def extract_content(self):
        print("Extracting content from contract...")

    def analyze_content(self):
        print("Analyzing contract content...")

    def save_results(self):
        print("Saving contract analysis results...")

# Client code
if __name__ == "__main__":
    report_processor = ReportProcessor()
    report_processor.process_document("report.docx")

    invoice_processor = InvoiceProcessor()
    invoice_processor.process_document("invoice.pdf")

    contract_processor = ContractProcessor()
    contract_processor.process_document("contract.doc")
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

OUTPUT

Loading report document...
Extracting content from report...
Analyzing report content...
Saving report analysis results...
Loading invoice document...
Extracting content from invoice...
Analyzing invoice content...
Saving invoice analysis results...
Loading contract document...
Extracting content from contract...
Analyzing contract content...
Saving contract analysis results...

CONCLUSION:

The code implements the Template Design Pattern to streamline the processing of various document types like reports, invoices, and contracts. An abstract superclass, **DocumentProcessor**, defines a template method, **process_document()**, encapsulating the document processing algorithm. Concrete subclasses, such as **ReportProcessor**, **InvoiceProcessor**, and **ContractProcessor**, extend **DocumentProcessor**, providing specific implementations for loading, extracting, analyzing, and saving document content. Client code demonstrates the instantiation of subclass instances and invocation of the processing method, ensuring consistent behavior across different document types while allowing customization for specific processing requirements. This design promotes code reusability, maintainability, and consistency by enforcing a common processing structure while accommodating variations in document processing logic.

REFERENCES:

[1] https://www.geeksforgeeks.org/template-method-design-pattern/