OOAD

Assignment 2 : Design Patterns

Name : Mohul Y P

SRN : PES1UG22CS360

Design patterns considered:

1. Flyweight Pattern (Blue Dashed Outlines)
   Purpose: To efficiently share word types and styles across multiple words

1. Chain of Responsibility Pattern (Green Dashed Outline) Purpose: To create a processing pipeline for formatting words

2. Factory Pattern (Orange Dashed Outline)
   Purpose: To centralize object creation and support the flyweight implementation

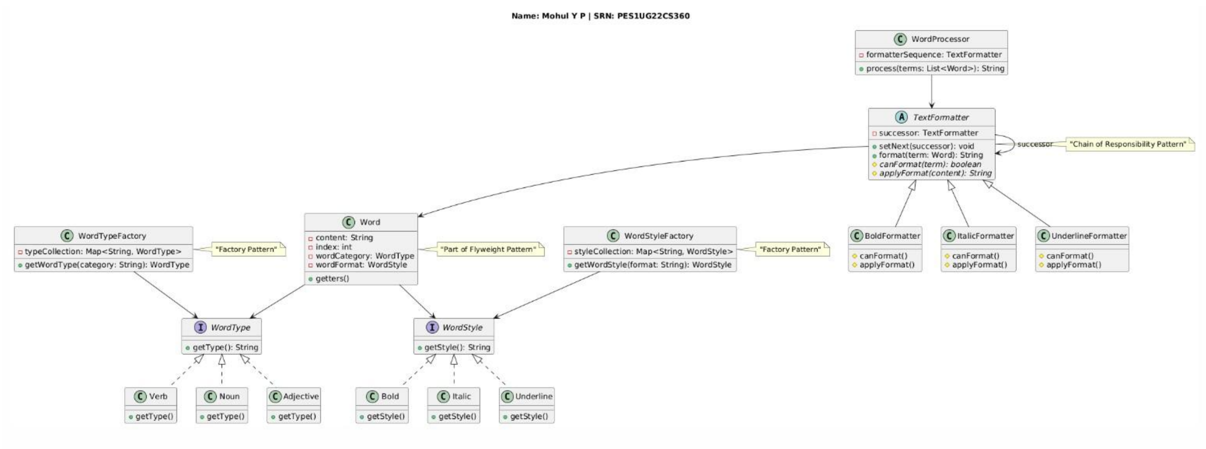
Design patterns used:

1. Flyweight Pattern (Blue Dashed Outlines)
   Purpose: To efficiently share word types and styles across multiple words

2. Chain of Responsibility Pattern (Green Dashed Outline) Purpose: To create a processing pipeline for formatting words

3. Factory Pattern (Orange Dashed Outline)
   Purpose: To centralize object creation and support the flyweight implementation

UML Diagram with Design Pattern highlighted:



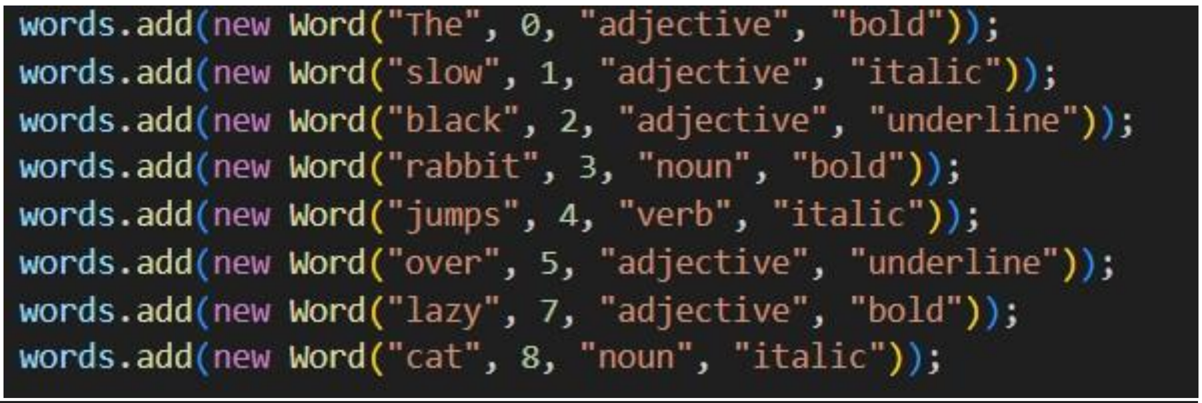
Code:

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List; import
java.util.Map;

// WordType Interface - Part of Flyweight Pattern
interface WordType {
   String getType();
}

// Concrete WordType implementations
class Verb implements WordType {     public
String getType() {
       return "verb";
   }
}

class Noun implements WordType {
   public String getType() {
return "noun";
   }
}

class Adjective implements WordType {     public
String getType() {
       return "adjective";
   }
}

// WordStyle Interface - Part of Flyweight Pattern
interface WordStyle {
   String getStyle();
}

// Concrete WordStyle implementations
```

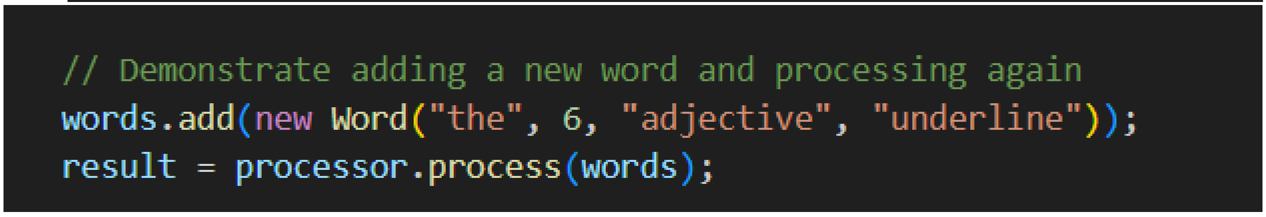
```
class Bold implements WordStyle {
public String getStyle() {        return
"bold";
   }
}

class Italic implements WordStyle {
   public String getStyle() {
return "italic";
   }
}

class Underline implements WordStyle {
public String getStyle() {        return
"underline";
   }
}

// WordTypeFactory - Factory for creating WordType instances class
WordTypeFactory {
   private static final Map<String, WordType> typeCollection = new HashMap<>();

   public static WordType getWordType(String category) {
WordType wordCategory = typeCollection.get(category);

      if (wordCategory == null) {
switch (category) {
            case "verb":
               wordCategory = new Verb();
               break;
case "noun":
               wordCategory = new Noun();
               break;
case "adjective":
               wordCategory = new Adjective();
break;              default:
               throw new IllegalArgumentException("Unknown word category: " +
category);
         }
         typeCollection.put(category, wordCategory);
      }
```

```
        return wordCategory;
    }
}

// WordStyleFactory - Factory for creating WordStyle instances class
WordStyleFactory {
    private static final Map<String, WordStyle> styleCollection = new
HashMap<>();

    public static WordStyle getWordStyle(String format) {
WordStyle wordFormat = styleCollection.get(format);

        if (wordFormat == null) {
switch (format) {
case "bold":
                    wordFormat = new Bold();
                    break;
case "italic":
                    wordFormat = new Italic();
                    break;
case "underline":
                    wordFormat = new Underline();
break;                  default:
                    throw new IllegalArgumentException("Unknown word format: " + format);
            }
            styleCollection.put(format, wordFormat);
        }

        return wordFormat;
    }
}

// Word class - Uses flyweight pattern for WordType and WordStyle
class Word {     private String content;     private int index;     private
WordType wordCategory;
    private WordStyle wordFormat;

    public Word(String content, int index, String category, String format) {
this.content = content;
        this.index = index;
        this.wordCategory = WordTypeFactory.getWordType(category);
this.wordFormat = WordStyleFactory.getWordStyle(format);
    }

    public String getText() {
        return content;
    }

    public int getPosition() {
return index;
    }

    public WordType getWordType() {
return wordCategory;
    }

    public WordStyle getWordStyle() {
        return wordFormat;
    }

    public String toString() {
        return "Word['" + content + "', position=" + index +
               ", type=" + wordCategory.getType() + ", style=" + wordFormat.getStyle() +
"]";
    }
}

// TextFormatter - Chain of Responsibility Pattern
abstract class TextFormatter {     private
TextFormatter successor;

    public void setNext(TextFormatter successor) {
        this.successor = successor;
    }

    public String format(Word term) {
        if (canFormat(term)) {
            return applyFormat(term.getText());
        } else if (successor != null) {
return successor.format(term);
        } else {
            return term.getText(); // No formatting applied
        }
    }

    protected abstract boolean canFormat(Word term);     protected
abstract String applyFormat(String content);
}

// Concrete TextFormatter implementations class
BoldFormatter extends TextFormatter {
    @Override
    protected boolean canFormat(Word term) {
        return term.getWordStyle().getStyle().equals("bold");
    }

    @Override
    protected String applyFormat(String content) {
        return "<b>" + content + "</b>";
    }
}

class ItalicFormatter extends TextFormatter {
    @Override
    protected boolean canFormat(Word term) {
        return term.getWordStyle().getStyle().equals("italic");
    }

    @Override
    protected String applyFormat(String content) {
        return "<i>" + content + "</i>";
    }
}

class UnderlineFormatter extends TextFormatter {
    @Override
    protected boolean canFormat(Word term) {
        return term.getWordStyle().getStyle().equals("underline");
    }

    @Override
    protected String applyFormat(String content) {
        return "<u>" + content + "</u>";
    }
}

// WordProcessor - Client class that uses formatters class
WordProcessor {
    private TextFormatter formatterSequence;

    public WordProcessor() {
        // Set up the chain of responsibility
        BoldFormatter boldHandler = new BoldFormatter();
        ItalicFormatter italicHandler = new ItalicFormatter();
        UnderlineFormatter underlineHandler = new UnderlineFormatter();

        boldHandler.setNext(italicHandler);
        italicHandler.setNext(underlineHandler);

        formatterSequence = boldHandler;
    }

    public String process(List<Word> terms) {
        // Sort words by position
        terms.sort(Comparator.comparingInt(Word::getPosition));

        StringBuilder output = new StringBuilder();
        for (Word term : terms) {
            String formatted = formatterSequence.format(term);
output.append(formatted).append(" ");
        }

        return output.toString().trim();
    }
}

// Main class for testing public class WordProcessingSystem {
public static void main(String[] args) {         // Create words with
different types and styles         List<Word> terms = new
ArrayList<>();         terms.add(new Word("The", 0, "adjective",
"bold"));         terms.add(new Word("slow", 1, "adjective",
"italic"));         terms.add(new Word("black", 2, "adjective",
"underline"));         terms.add(new Word("rabbit", 3, "noun",
"bold"));        terms.add(new Word("jumps", 4, "verb", "italic"));
terms.add(new Word("over", 5, "adjective", "underline"));
terms.add(new Word("lazy", 7, "adjective", "bold"));
        terms.add(new Word("cat", 8, "noun", "italic"));

        // Process the words
        WordProcessor processor = new WordProcessor();
        String output = processor.process(terms);

        System.out.println("Processed sentence: " + output);

        // Demonstrate adding a new word and processing again
terms.add(new Word("the", 6, "adjective", "underline"));        output
= processor.process(terms);

        System.out.println("\nUpdated sentence: " + output);
    }
}
```

Input:

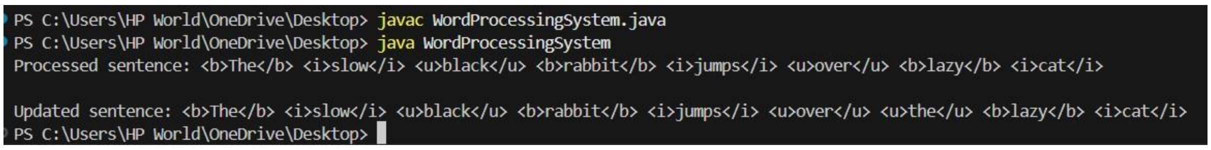


```
words.add(new Word("The", 0, "adjective", "bold"));
words.add(new Word("slow", 1, "adjective", "italic"));
words.add(new Word("black", 2, "adjective", "underline"));
words.add(new Word("rabbit", 3, "noun", "bold"));
words.add(new Word("jumps", 4, "verb", "italic"));
words.add(new Word("over", 5, "adjective", "underline"));
words.add(new Word("lazy", 7, "adjective", "bold"));
words.add(new Word("cat", 8, "noun", "italic"));
```



```
// Demonstrate adding a new word and processing again
words.add(new Word("the", 6, "adjective", "underline"));
result = processor.process(words);
```

Output:



```
PS C:\Users\HP World\OneDrive\Desktop> javac WordProcessingSystem.java
PS C:\Users\HP World\OneDrive\Desktop> java WordProcessingSystem
Processed sentence: <b>The</b> <i>slow</i> <u>black</u> <b>rabbit</b> <i>jumps</i> <u>over</u> <b>lazy</b> <i>cat</i>

Updated sentence: <b>The</b> <i>slow</i> <u>black</u> <b>rabbit</b> <i>jumps</i> <u>over</u> <u>the</u> <b>lazy</b> <i>cat</i>
PS C:\Users\HP World\OneDrive\Desktop> 
```