(CSE336s) Software Design Patterns

Omar Mamon Hamed (2100767)

Project Documentation

1) List of the Program

Feel free to check out my repository

- **Language Used: Java**
- **Classes and Interfaces:**

I. Builder Interface:
`TextConverter`

  - Defines the methods to handle text conversion tasks (`ConvertCharacter`, `ConvertFontChange`, and `ConvertParagraph`).

II. Concrete Builders:

  - **`ASCIIConverter`**: Produces plain ASCII text as output.
  - **`TeXConverter`**: Produces LaTeX-style formatted text.
  - **`TextWidgetConverter`**: Produces a textual representation of styled widgets for text formatting.

III. Director (or Client):
`RTFReader`

  - Parses the RTF-like input and calls the appropriate builder methods to construct the output.

IV. Product:
The generated outputs:

  - ASCII text (from `ASCIIConverter`)
  - LaTeX-formatted text (from `TeXConverter`)
  - Styled widget-like text representation (from `TextWidgetConverter`).

V. Client Code:
`Main`

  - The client initializes the builders and the `RTFReader`, then demonstrates the building process for each type of output.

- **Source code:**

I. FontStyle.java

```java
package rtf;

public enum FontStyle {
    BOLD, ITALIC, UNDERLINE
}
```

II. TextConverter.java

```java
package rtf;

public interface TextConverter {
    void ConvertCharacter(char c);
    void ConvertFontChange(FontStyle font);
    void ConvertParagraph();
}
```

III.RTFReader.java

```java
package rtf;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RTFReader {
    private TextConverter builder;
    private static final Pattern TOKEN_PATTERN =
Pattern.compile("\{(char|font|par)(?::([^}]+))?\}");

    public RTFReader(TextConverter builder) {
        this.builder = builder;
    }

    public void ParseRTF(String rtf_input) {
        Matcher matcher = TOKEN_PATTERN.matcher(rtf_input);
        int lastIndex = 0;

        while (matcher.find()) {
            String tokenType = matcher.group(1);
            String tokenValue = matcher.group(2);

            switch (tokenType) {
                case "char":
                    if (tokenValue != null && tokenValue.length() == 1) {
                        builder.ConvertCharacter(tokenValue.charAt(0));
                    }
                    break;
                case "font":
                    if (tokenValue != null) {
                        switch (tokenValue) {
                            case "bold":
                                builder.ConvertFontChange(FontStyle.BOLD);
                                break;
                            case "italic":
                                builder.ConvertFontChange(FontStyle.ITALIC);
                                break;
                            case "underline":
                                builder.ConvertFontChange(FontStyle.UNDERLINE);
                                break;
                        }
                    }
                    break;
                case "par":
                    builder.ConvertParagraph();
                    break;
            }

            lastIndex = matcher.end();
        }

        // Handle any parsing errors or unexpected input
        if (lastIndex < rtf_input.length()) {
            System.err.println("Warning: Unparsed input remains: " +
                rtf_input.substring(lastIndex));
        }
    }
}
```

IV. ASCIIConverter.java

```java
package rtf;

public class ASCIIConverter implements TextConverter {
    private StringBuilder result = new StringBuilder();

    @Override
    public void ConvertCharacter(char c) {
        result.append(c);
    }

    @Override
    public void ConvertFontChange(FontStyle font) {
        // ASCII doesn't support font changes
    }

    @Override
    public void ConvertParagraph() {
        result.append("n");
    }

    public String GetASCIIText() {
        return result.toString();
    }
}
```

V. TeXConverter.java

```java
package rtf;

import java.util.EnumSet;

public class TeXConverter implements TextConverter {
    private StringBuilder result = new StringBuilder();
    private EnumSet<FontStyle> activeStyles = EnumSet.noneOf(FontStyle.class);

    @Override
    public void ConvertCharacter(char c) {
        String convertedChar = escapeTeXSpecialChar(c);

        // Apply active styles
        if (!activeStyles.isEmpty()) {
            convertedChar = wrapInTeXCommands(convertedChar);
        }

        result.append(convertedChar);
    }

    @Override
    public void ConvertFontChange(FontStyle font) {
        if (activeStyles.contains(font)) {
            activeStyles.remove(font);
        } else {
            activeStyles.add(font);
        }
    }

    @Override
    public void ConvertParagraph() {
        result.append("nn");
    }

    private String escapeTeXSpecialChar(char c) {
        switch (c) {
            case '&': return "\&";
            case '%': return "\%";
            case '$': return "\$";
            case '#': return "\#";
            case '_': return "\_";
            case '{': return "\{";
            case '}': return "\}";
            default: return String.valueOf(c);
        }
    }

    private String wrapInTeXCommands(String text) {
        if (activeStyles.containsAll(EnumSet.of(FontStyle.BOLD, FontStyle.ITALIC))) {
            return "\textbf{\textit{" + text + "}}";
        }
        if (activeStyles.contains(FontStyle.BOLD)) {
            return "\textbf{" + text + "}";
        }
        if (activeStyles.contains(FontStyle.ITALIC)) {
            return "\textit{" + text + "}";
        }
        if (activeStyles.contains(FontStyle.UNDERLINE)) {
            return "\underline{" + text + "}";
        }
        return text;
    }

    public String GetTeXText() {
        return result.toString();
    }
}
```

VI. TextWidgetConverter.java

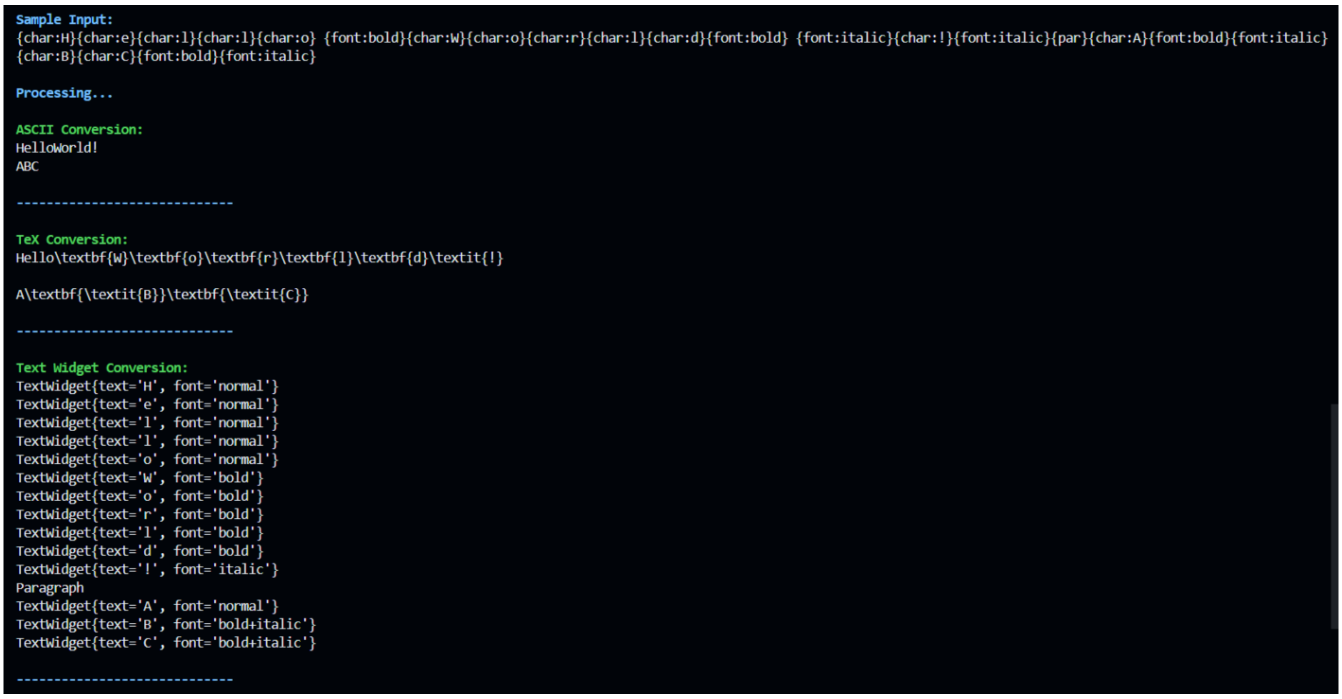
```
package rtf;

import java.util.ArrayList;
import java.util.EnumSet;
import java.util.List;

public class TextWidgetConverter implements TextConverter {
    private List<String> elements = new ArrayList<>();
    private EnumSet<FontStyle> activeStyles = EnumSet.noneOf(FontStyle.class);

    @Override
    public void ConvertCharacter(char c) {
        // Create a TextWidget for each character with current font styles
        String stylesStr = activeStyles.isEmpty() ? "normal" :
            activeStyles.stream()
                .map(Enum::name)
                .reduce((s1, s2) -> s1 + "+" + s2)
                .orElse("normal");

        elements.add(String.format("TextWidget{text='%s', font='%s'}",
            c, stylesStr.toLowerCase()));
    }

    @Override
    public void ConvertFontChange(FontStyle font) {
        // Toggle the font style for subsequent characters
        if (activeStyles.contains(font)) {
            activeStyles.remove(font);
        } else {
            activeStyles.add(font);
        }
```

```java
    }

    @Override
    public void ConvertParagraph() {
        // Add paragraph separator
        elements.add("Paragraph");
    }

    public String GetTextWidget() {
        return String.join("n", elements);
    }
}
```

VII. Main.java

```java
import rtf.*;

public class Main {
    public static void main(String[] args) {
        // Enhanced sample input with more complex font toggling
        String sampleInput =
            "{char:H}{char:e}{char:l}{char:l}{char:o} " +
            "{font:bold}{char:W}{char:o}{char:r}{char:l}{char:d}{font:bold} " +
            "{font:italic}{char:!}{font:italic}{par}" +

"{char:A}{font:bold}{font:italic}{char:B}{char:C}{font:bold}{font:italic}";

        // Print the sample input for visualization
        System.out.println("n33[1;34mSample Input:33[0m");
        System.out.println(sampleInput);
        System.out.println("n33[1;34mProcessing...33[0mn");

        // Demonstrate ASCII Conversion
        System.out.println("33[1;32mASCII Conversion:33[0m");
        TextConverter asciiConverter = new ASCIIConverter();
        RTFReader asciiReader = new RTFReader(asciiConverter);
        asciiReader.ParseRTF(sampleInput);
        System.out.println(((ASCIIConverter) asciiConverter).GetASCIIText());
        System.out.println("n33[1;34m-----------------------------33[0mn");

        // Demonstrate TeX Conversion
        System.out.println("33[1;32mTeX Conversion:33[0m");
        TextConverter texConverter = new TeXConverter();
        RTFReader texReader = new RTFReader(texConverter);
        texReader.ParseRTF(sampleInput);
        System.out.println(((TeXConverter) texConverter).GetTeXText());
        System.out.println("n33[1;34m-----------------------------33[0mn");

        // Demonstrate Text Widget Conversion
        System.out.println("33[1;32mText Widget Conversion:33[0m");
        TextConverter widgetConverter = new TextWidgetConverter();
        RTFReader widgetReader = new RTFReader(widgetConverter);
        widgetReader.ParseRTF(sampleInput);
        System.out.println(((TextWidgetConverter) widgetConverter).GetTextWidget());
        System.out.println("n33[1;34m-----------------------------33[0mn");
     }
}
```

2) Output Snapshot



```
Sample Input:
{char:H}{char:e}{char:l}{char:l}{char:o} {font:bold}{char:W}{char:o}{char:r}{char:l}{char:d}{font:bold} {font:italic}{char:!}{font:italic}{par}{char:A}{font:bold}{font:italic}
{char:B}{char:C}{font:bold}{font:italic}

Processing...

ASCII Conversion:
HelloWorld!
ABC

----------------------------

TeX Conversion:
Hello\textbf{W}\textbf{o}\textbf{r}\textbf{l}\textbf{d}\textit{!}

A\textbf{\textit{B}}\textbf{\textit{C}}

----------------------------

Text Widget Conversion:
TextWidget{text='H', font='normal'}
TextWidget{text='e', font='normal'}
TextWidget{text='l', font='normal'}
TextWidget{text='l', font='normal'}
TextWidget{text='o', font='normal'}
TextWidget{text='W', font='bold'}
TextWidget{text='o', font='bold'}
TextWidget{text='r', font='bold'}
TextWidget{text='l', font='bold'}
TextWidget{text='d', font='bold'}
TextWidget{text='!', font='italic'}
Paragraph
TextWidget{text='A', font='normal'}
TextWidget{text='B', font='bold+italic'}
TextWidget{text='C', font='bold+italic'}

----------------------------
```

3) Discussion

I.FontStyle Enum

- Represents the various font styles (e.g., **BOLD**, *ITALIC*, UNDERLINE) used in the program.
- Provides a simple way to manage font changes during text processing.

II.TextConverter Interface

- Acts as the **abstract builder** in the Builder Design Pattern.
- Defines the methods that all concrete builders must implement:
  - convertCharacter(char c): Converts a character into the desired format.
  - convertFontChange(FontStyle font): Applies font style changes.
  - convertParagraph(): Handles paragraph breaks.

III.ASCIIConverter Class

- Implements the TextConverter interface to create plain ASCII text output.
- Handles characters, font changes, and paragraphs in a format suitable for simple text representation.
- **Example: Converts {char:H} into H and ignores font styling**.

IV.TeXConverter Class

- Implements the TextConverter interface to generate TeX (LaTeX) formatted output.
- Wraps font changes using TeX commands like \textbf for bold and \textit for italic.
- ***Example: Converts {font:bold}{char:W} into \textbf{W}.***

V.TextWidgetConverter Class

- Implements the TextConverter interface to create a structured text representation, mimicking GUI-style text widgets.
- Processes each character and font style change into TextWidget objects for further use in UI-related applications.
- ***Example: Converts {char:A}{font:bold} into TextWidget{text='A', font='bold'}.***

VI.RTFReader Class

- Acts as the director in the Builder Design Pattern.
- Reads the input stream formatted in an RTF-like structure (e.g., {char:X}{font:bold}) and delegates the parsing and conversion to a TextConverter.
- Ensures the correct sequence of builder method calls (convertCharacter, convertFontChange, convertParagraph) based on the input.
- ***Example: Parses {char:H}{char:e} and calls convertCharacter('H') and convertCharacter('e') on the assigned TextConverter.***

VII.Main Class

- Serves as the client in the Builder Design Pattern.
- Demonstrates the use of RTFReader with different TextConverter implementations.
- ***Example: Initializes ASCIIConverter, TeXConverter, and TextWidgetConverter, and passes them to RTFReader to generate outputs for each format.***