Subrata University

College of Information Technology Al Jamail

Smart Alternative Writing Systems Generator

Mini Report – Software Design Patterns

يلال عبدالله عبيد & عبدالله رجب حسن & محمد عمر التائب

1. Introduction

This project implements a **Smart Alternative Writing Systems Generator**, a Java-based application that converts plain text into alternative writing systems such as Morse Code, Braille, and a Custom Secret Cipher. The system also supports decoding, allowing encoded text to be converted back to plain text. The project demonstrates the practical application of three core design patterns—one from each main category (Creational, Structural, Behavioral)—as required by the assignment. A simple graphical user interface (GUI) built with Java Swing provides an intuitive way for users to interact with the system.

2. Design Patterns Applied

2.1 Prototype Pattern (Creational)

The **Prototype pattern** was used to manage the creation of encoder objects efficiently. Instead of instantiating new encoder objects from scratch each time, the system **clones existing prototype instances**.

- **Implementation:**
  The abstract class `EncoderPrototype` defines the common interface and implements the `clone()` method. Concrete encoders (`MorseEncoder`, `BrailleEncoder`, `SecretEncoder`) extend this class and provide their own mapping between characters and symbols.
- **Why Prototype?**
  This approach reduces object-creation overhead, ensures consistency across encoder instances, and aligns with the Creational pattern requirement.

2.2 Proxy Pattern (Structural)

The **Proxy pattern** was implemented to control access to certain encoders based on user roles.

- **Implementation:**
  The `EncoderProxy` class acts as a protection proxy. It checks the user's role (Student or Researcher) before granting access to the Secret Cipher encoder. Students are prevented from using the Secret Cipher, while Researchers have full access.
- **Why Proxy?**
  This pattern separates access-control logic from the core encoding

functionality, enhancing security without modifying the encoder classes themselves.

2.3 Strategy Pattern (Behavioral)

The **Strategy pattern** was applied to encapsulate different methods of displaying encoded output.

- **Implementation:**
  The `DisplayStrategy` interface defines a common `display()` method. Two concrete strategies are provided:
  - `CompactStrategy`: shows only the encoded symbols.
  - `VerboseStrategy`: shows each original character alongside its encoded form.
- **Why Strategy?**
  This allows the display behavior to be changed dynamically at runtime, adhering to the Open/Closed Principle and fulfilling the Behavioral pattern requirement.

3. System Features & User Interaction

The system offers the following key functionalities:

- **Text Encoding:** Convert input text to Morse, Braille, or Secret Cipher.
- **Text Decoding:** Convert encoded text back to plain text (for Morse and Braille).
- **Role-Based Access:** Only users with the “Researcher” role can use the Secret Cipher.
- **Display Modes:** Users can toggle between Compact and Verbose output formats.

The GUI provides input fields, selection boxes, and buttons to control these features, making the system easy to use without requiring command-line interactions.

4. Why These Patterns Were Chosen

- **Prototype:** Optimal for systems where object creation is costly and multiple similar instances are needed.
- **Proxy:** Ideal for adding security layers without altering business logic.

- **Strategy:** Perfect for enabling flexible, swappable algorithms (here, display methods).

Together, these patterns ensure the system is **modular, maintainable, and extensible**.

5. Conclusion

This project successfully applies three fundamental design patterns to build a functional and educational software system. The Prototype, Proxy, and Strategy patterns each address distinct design concerns—object creation, access control, and behavioral flexibility—while working together cohesively. The implementation satisfies all assignment requirements and illustrates how design patterns can be used to create clean, scalable, and professional-grade Java applications.