The code exhibits several design patterns and principles. Here are some of the notable ones:

1. **Builder Pattern:**
   * The IfStatement class uses the @Data annotation, which, when used with Lombok, automatically generates getter, setter, and other boilerplate code.
   * This is similar to the Builder pattern, where a separate builder class is used to construct an object with many optional parameters.
2. **Composite Pattern:**
   * The IfStatement class represents a tree structure where each node (IfStatement) can have child nodes (elseStatement). This structure allows for the representation of nested if-else statements.
3. **Strategy Pattern:**
   * The evaluateIf method uses a strategy-like pattern to handle different comparison operations (==, >, <). It encapsulates the algorithm for each condition type, similar to the strategy pattern.
4. **Chain of Responsibility Pattern:**
   * The evaluateIfElseJson method uses a loop to iterate through conditions, and if one condition fails, it moves to the next condition or the else block. This is similar to the Chain of Responsibility pattern, where each condition represents a link in the chain, and the responsibility is passed along the chain until it is handled.
5. **Factory Method Pattern:**
   * The ObjectMapper class is used to create instances of IfStatement from JSON. This is similar to the Factory Method pattern, where a factory method is responsible for creating objects.
6. **Exception Handling:**
   * The code uses exception handling to ignore different data types in the evaluateIf method. While not a design pattern per se, it follows good exception-handling practices.
7. **Single Responsibility Principle (SRP):**
   * Methods such as evaluateIfElseJson, evaluateCondition, getVariableValue, etc., seem to adhere to the Single Responsibility Principle by having a clear and specific purpose.

These patterns and principles contribute to the maintainability, readability, and flexibility of the code.