To build the calculator, we applied clean code principles and structured the project into four components: Tokenizer, Token, ExpressionParser, and Calculator. The tokenizer breaks the input string into tokens for easier parsing. We used recursive descent parsing to evaluate expressions with correct operator precedence (* before +/-). The grammar supports numbers, parentheses, and unary negation. We chose this strategy for its readability, modularity, and ease of extension (e.g., to add / or ^). The parser handles expressions like -3 + (2 * 4) naturally. Error handling ensures malformed expressions are caught early. We assumed the input is syntactically valid and limited scope to arithmetic expressions without variables or functions. The design prioritizes testability, readability, and respect for SOLID principles. Bonus features (parentheses and negative numbers) are already integrated thanks to the grammar structure.