Task 1: Identified Code Smells

1. Long Method/Class - UI.java
   Location: UI.java, actionPerformed method and UI constructor
   Problem: The actionPerformed method is very long, over 100 lines, with repetitive if-else conditions. The UI class handles too many responsibilities such as button creation, layout management, event handling, and business logic coordination. This violates the Single Responsibility Principle and makes the code hard to maintain and test.

2. Primitive Obsession - Calculator.java
   Location: Calculator.java, BiOperatorModes and MonoOperatorModes enums
   Problem: The calculator uses raw enums and double primitives without proper abstraction. This leads to repetitive conditional logic in methods like calculateBiImpl and calculateMono and makes it difficult to add new operations in the future.

3. Feature Envy - UI.java
   Location: UI.java, actionPerformed method
   Problem: The UI class contains too much knowledge about Calculator's internal modes and operations. It directly references operator modes and handles conversion logic that should be part of the Calculator class.

Task 2: Applied Design Patterns

Problem Identified: Lack of abstraction for operations
Files affected: Calculator.java and UI.java
Patterns applied: Strategy Pattern and Command Pattern
Refactored files: Operation.java (interface), BinaryOperation.java and UnaryOperation.java (abstract classes), AddOperation.java, MultiplyOperation.java, etc. (concrete operations), Calculator.java (refactored), UI.java (simplified)

Solution

1. Strategy Pattern: Created an Operation interface with a calculate method. Specific operations are implemented as separate classes. This removes long if-else chains in the Calculator class.
2. Factory Pattern: Created an OperationFactory to centralize the creation of operations.
3. Command Pattern: Created command objects for calculator functions, decoupling the UI from business logic.

Task 3: New Feature Implementation

Added memory functionality to store, recall, add to, and clear memory.
needed..

Pattern Applied: Observer Pattern

Location:Folder:operator

addOperator.java

calculatorEngine.java

OperationCommand.java

ResourseLoader.java

Applied Strategy Pattern

Folder: Refector ALL JAVA FIL

Notify UI when memory state changes

Benefits of Refactoring

1. Improved maintainability
2. Better testability.
3. Reduced coupling.
4. Enhanced extensibility
5. Cleaner code
6. Code smells are eliminated and SOLID principles are followed.