Examination

Architecture Overview

The system is structured around a Facade (MathPlot) that coordinates multiple independent subsystems:

- Expression parsing
- Expression representation (AST)
- Differentiation
- Simplification
- Plotting
- Numerical integration (area calculation)

Each concern is implemented in a separate package

Facade Pattern

Regards MathPlot

MathPlot provides a single, public API for the entire system, hiding internal complexity from the GUI and tests.

Accept user input (expression + format)
Coordinate parsing, differentiation, plotting, printing, and area calculation
Expose simplified methods such as:

- setExpression(...)
- plot(...)
- area(...)
- print(...)

Strategy Pattern

Regards Parsing :
ExpressionParser (strategy interface)
AOSParserAdapter
RPNParserAdapter

Parsing algorithms vary depending on the input format (AOS or RPN).
Each parser is implemented as a separate strategy, adapting the provided legacy parsers to a common interface.

Adding a new expression format does not require changes to existing parsing logic or MathPlot

Composite Pattern

Regards Expression tree :
Expr (component)
Const, Var (leaf nodes)
Unary, Binary (composite nodes)

Mathematical expressions are represented as a recursive tree structure.

Strategy Pattern

Regards Plotting:
PlotStrategy

CartesianPlotStrategy
PolarPlotStrategy

Different plot types (Cartesian and Polar) are implemented as independent strategies.
Each strategy knows how to:

- Draw its grid
- Draw reference axes
- Plot the function and its derivative

Adding a new plot type does not require changes to existing plots
MathPlot selects the strategy at runtime

Iterator Pattern

Regarding these classes:
Point.Iterator
SampleIterator
PolarIterator

Sampling of function values is abstracted via iterators.
Plotting code does not depend on how points are generated.

- Supports continuous functions
- Allows future support for discontinuities via hasBreak()
- Decouples plotting from numerical sampling

Simplification

Simplification does not affect evaluation or differentiation logic
Expressions remain valid whether simplified or not

Encapsulation

Regarding the Area Calculation
AreaCalculator
AreaType (Rectangular, Trapezoidal)

Adding a new integration method does not affect existing code structure
Algorithms are isolated from plotting and parsing

Adapter Pattern

Regarding
MathPlot.Parsers.AOS
MathPlot.Parsers.RPN

provided parsers are adapted to the system using adapters.

Integration of Provided Code

The plotting component (`Plotter`) supplied with the assignment was integrated without modification to its internal logic or behavior.
All rendering, coordinate transformation, and mouse interaction (panning and zooming) functionality remains exactly as provided.

Modifications were strictly limited to the sections explicitly marked with `// YOU CAN CHANGE HERE`, where the plotting engine is used but not altered. The `Plotter` class continues to act as a self-contained rendering subsystem responsible solely for visual output.

This design ensures a clear separation of concerns:

- The provided `Plotter` handles rendering and user interaction
- The student-implemented code handles expression parsing, evaluation, differentiation, and plotting strategy selection

By preserving the original plotting infrastructure and extending functionality through well-defined interfaces and design patterns, the solution respects the assignment constraints while remaining modular and extensible.