The Strategy Pattern
Structure & Functionality

Douglas C. Schmidt
Learning Objectives in This Lesson

• Recognize how the *Strategy* pattern can be applied in the expression tree processing app to encapsulate variability of algorithm & platform behaviors via common APIs.

• Understand the structure & functionality of the *Strategy* pattern.
Strategy

Intent

- Define a family of algorithms, encapsulate each one, & make them interchangeable to let clients & algorithms vary independently

See en.wikipedia.org/wiki/Strategy_pattern
Applicability

- When an object should be configurable with one of many algorithms
**Applicability**

- When an object should be configurable with one of many algorithms

- *And* all algorithms can be encapsulated
**Applicability**

- When an object should be configurable with one of many algorithms
- *And* all algorithms can be encapsulated
- *And* one interface covers all encapsulations
Strategy

GoF Object Behavioral

Structure & participants

- Context
  - ContextInterface()

- Strategy
  - AlgorithmInterface()

- ConcreteStrategyA
  - AlgorithmInterface()

- ConcreteStrategyB
  - AlgorithmInterface()

- ConcreteStrategyC
  - AlgorithmInterface()
Structure & participants

Strategy

GoF Object Behavioral

Context

ContextInterface()

strategy

Strategy

AlgorithmInterface()

ConcreteStrategyA

AlgorithmInterface()

ConcreteStrategyB

AlgorithmInterface()

ConcreteStrategyC

AlgorithmInterface()

Pre_Order_ET_Iter_Impl, Post_Order_ET_Iter_Impl, Level_Order_ET_Iterator_Impl, In_Order_ET_Iter_Impl, etc.
Context is primarily useful if some strategies need more than the common API.
Strategy (object behavioral) closely resembles Bridge (object structural).
These classes play very different roles.

*Strategy* (object behavioral) closely resembles *Bridge* (object structural).