The Composite Pattern
Implementation in C++

Douglas C. Schmidt
Learning Objectives in This Lesson

- Recognize how the *Composite* pattern can be applied to make the expression tree more uniform & extensible.
- Understand the structure & functionality of the *Composite* pattern.
- Know how to implement the *Composite* pattern in C++. 
Douglas C. Schmidt

Implementing the Composite Pattern in C++
Composite example in C++

- Build an expression tree based on recursively composed objects.

```
Component_Node *11 =
    new Leaf_Node(5);
Component_Node *12 =
    new Leaf_Node(3);
Component_Node *13 =
    new Leaf_Node(4);
```
Composite example in C++

• Build an expression tree based on recursively composed objects.

Component_Node *l1 =
    new Leaf_Node(5);
Component_Node *l2 =
    new Leaf_Node(3);
Component_Node *l3 =
    new Leaf_Node(4);
Component_Node *u1 =
    new Composite_Negate_Node(l1);
Component_Node *b1 =
    new Composite_Add_Node(l2, l3);
Composite example in C++

- Build an expression tree based on recursively composed objects.

```cpp
Component_Node *l1 = new Leaf_Node(5);
Component_Node *l2 = new Leaf_Node(3);
Component_Node *l3 = new Leaf_Node(4);
Component_Node *u1 = new Composite_Negate_Node(l1);
Component_Node *b1 = new Composite_Add_Node(l2, l3);
Component_Node *b2 = new Composite_Multiply_Node(u1, b1);
```
Composite example in C++

- Build an expression tree based on recursively composed objects.

```cpp
Component_Node *l1 = new Leaf_Node(5);
Component_Node *l2 = new Leaf_Node(3);
Component_Node *l3 = new Leaf_Node(4);
Component_Node *u1 = new Composite_Negate_Node(l1);
Component_Node *b1 = new Composite_Add_Node(l2, l3);
Component_Node *b2 = new Composite_Multiply_Node(u1, b1);
```

*C++ lacks garbage collector, so this implementation requires intentional deallocation of memory (e.g., via reference counting) when it’s no longer needed.*

Composite example in C++

- Build an expression tree based on recursively composed objects.

```cpp
unique_ptr<Component_Node> expr_tree
    = make_expression_tree("-5 * (3 + 4)");
```

A better way to build an expression tree is to apply a Creational pattern that shields client programs from the details of how the composite expression tree is implemented.

See upcoming lessons on the **Builder** (\& **Interpreter**) patterns.