Overview of the Expression Tree Processing App Case Study (Part 2)

Douglas C. Schmidt

Learning Objectives in This Lesson

- Understand the goals of the object-oriented (OO) expression tree case study.
- Recognize the key behavioral & structural properties in the expression tree domain.
- Evaluate the functional & nonfunctional requirements of the case study.
 Non-runtime qualities



Patterns are best applied to address requirements, rather than applied blindly!

Learning Objectives in This Lesson

- Understand the goals of the object-oriented (OO) expression tree case study.
- Recognize the key behavioral & structural properties in the expression tree domain.
- Evaluate the functional & nonfunctional requirements of the case study.
- Put all the pieces together.



Douglas C. Schmidt

Functional & Non-Functional Requirements of the Case Study

Functional & Non-Functional Requirements

 A functional requirement defines what a system should be able to do, i.e., the behavior it should perform.



en.wikipedia.org/wiki/Functional_requirements has more information.

Functional & Non-Functional Requirements

• A **non-functional requirement** defines specific criteria that can be used to judge the operation of a system, rather than its specific behaviors.



 Non-Functional requirements are also called "quality attributes" of a system.

en.wikipedia.org/wiki/Non-functional_requirement has more information.

 "Succinct mode"—the calculator interface evaluates arithmetic expressions input by a user that must conform to a grammar

```
expr ::= factor expr-tail
expr-tail ::= add sub expr
              | /* empty */;
factor ::= term factor-tail
factor-tail ::= mul div factor
                | /* empty */;
mul_div ::= 'x' | '/'
add sub ::= '+' | '-'
term :: NUMBER | '(' expr ')'
```

- The succinct mode can be command-line or GUI interface.
 - In the GUI version, a user presses Android buttons to enter expressions.



- The succinct mode can be command-line or GUI interface.
 - In the GUI version, a user presses Android buttons to enter expressions.
 - In the command-line version a user designates input expressions via various notations.
 - e.g., in-fix, post-fix, etc.





In-fix expressions can contain parenthesized sub-expressions.

- "Verbose mode"—prompts the user to enter command requests that control app behavior
 - The order of these command requests must follow a specific protocol.

Command	Behavior
format	Allows the user to select the input format (e.g., in-fix, post-fix, etc.)
expr	Allows the user to designate the current input expression
set	Sets a variable that can be used in an expression
print	Prints the current input expression using the designated traversal order (e.g., in-order, post-order, pre-order, etc.)
eval	Evaluates the value of the current input expression
quit	Exits the program

- "Verbose mode"—prompts the user to enter command requests that control app behavior
 - The order of these command requests must follow a specific protocol.

Command	Behavior
format	Allows the user to select the input format (e.g., in-fix, post-fix, etc.)
expr	Allows the user to designate the current input expression
set	Sets a variable that can be used in an expression
print	Prints the current input expression using the designated traversal order (e.g., in-order, post-order, pre-order, etc.)
eval	Evaluates the value of the current input expression
quit	Exits the program

- "Verbose mode"—prompts the user to enter command requests that control app behavior
 - The order of these command requests must follow a specific protocol.

Command	Behavior
format	Allows the user to select the input format (e.g., in-fix, post- fix, etc.)
expr	Allows the user to designate the current input expression
set	Sets a variable that can be used in an expression
print	Prints the current input expression using the designated traversal order (e.g., in-order, post-order, pre-order, etc.)
eval	Evaluates the value of the current input expression
quit	Exits the program

- "Verbose mode"—prompts the user to enter command requests that control app behavior
 - The order of these command requests must follow a specific protocol.

Command	Behavior
format	Allows the user to select the input format (e.g., in-fix, post-fix, etc.)
expr	Allows the user to designate the current input expression
set	Sets a variable that can be used in an expression
print	Prints the current input expression using the designated traversal order (e.g., in-order, post-order, pre-order, etc.)
eval	Evaluates the value of the current input expression
quit	Exits the program

- "Verbose mode"—prompts the user to enter command requests that control app behavior
 - The order of these command requests must follow a specific protocol.

Command	Behavior
format	Allows the user to select the input format (e.g., in-fix, post-fix, etc.)
expr	Allows the user to designate the current input expression
set	Sets a variable that can be used in an expression
print	Prints the current input expression using the designated traversal order (e.g., in-order, post-order, pre-order, etc.)
eval	Evaluates the value of the current input expression
quit	Exits the program

- "Verbose mode"—prompts the user to enter command requests that control app behavior
 - The order of these command requests must follow a specific protocol.

Command	Behavior
format	Allows the user to select the input format (e.g., in-fix, post-fix, etc.)
expr	Allows the user to designate the current input expression
set	Sets a variable that can be used in an expression
print	Prints the current input expression using the designated traversal order (e.g., in-order, post-order, pre-order, etc.)
eval	Evaluates the value of the current input expression
quit	Exits the program

- "Verbose mode"—prompts the user to enter command requests that control app behavior
 - The order of these command requests must follow a specific protocol.

Command	Behavior
format	Allows the user to select the input format (e.g., in-fix, post-fix, etc.)
expr	Allows the user to designate the current input expression
set	Sets a variable that can be used in an expression
print	Prints the current input expression using the designated traversal order (e.g., in-order, post-order, pre-order, etc.)
eval	Evaluates the value of the current input expression
quit	Exits the program

- The verbose mode can be accessed via:
 - A GUI interface



- The verbose mode can be accessed via:
 - A GUI interface
 - A command-line interface

```
expression_tree
 "D:\Douglas Schmidt\Dropbox\Documents\Vandy\cs251\CPlusPlus\ex
1a. format [in-order]
1b. set [variable=value]
2. expr [expression]
3a. eval [post-order]
3b. print [in-order | pre-order | post-order | level-order]
0. quit
>format in-order
1. expr [expression]
2a. eval [post-order]
 2b. print [in-order | pre-order | post-order | level-order]
Oa. format [in-order]
Ob. set [variable=value]
Oc. quit
>expr -5 * (3 + 4)
```



• Apply a pattern-oriented OO design to simplify extensibility & portability



- Apply a pattern-oriented OO design to simplify extensibility & portability, e.g.,
 - Add new operations on the expression tree nodes without modifying the tree structure or implementation



- Apply a pattern-oriented OO design to simplify extensibility & portability, e.g.,
 - Add new operations on the expression tree nodes without modifying the tree structure or implementation, e.g.,
 - Print the contents of the expression tree in various traversal orders



- "In-order" traversal = -5x (3+4)
- "Pre-order" traversal = x-5+34
- "Post-order" traversal = 5-34+×
- "Level-order" traversal = **x**-+534

- Apply a pattern-oriented OO design to simplify extensibility & portability, e.g.,
 - Add new operations on the expression tree nodes without modifying the tree structure or implementation, e.g.,
 - Print the contents of the expression tree in various traversal orders
 - Compute the "value" of the expression tree
 - e.g., via a post-order traversal & stack-based evaluator



1.S = [5]	<pre>push(node.item())</pre>
2. S = [-5]	<pre>push(-pop())</pre>
3.S = [-5, 3]	<pre>push(node.item())</pre>
4. S = [-5, 3, 4]	<pre>push(node.item())</pre>
5.S = [-5, 7]	<pre>push(pop()+pop())</pre>
6. S = [-35]	<pre>push(pop()*pop())</pre>

- Apply a pattern-oriented OO design to simplify extensibility & portability, e.g.,
 - Add new operations on the expression tree nodes without modifying the tree structure or implementation, e.g.,
 - Print the contents of the expression tree in various traversal orders
 - Compute the "value" of the expression tree
 - Perform semantic analysis & optimization, generate code, etc.



- Apply a pattern-oriented OO design to simplify extensibility & portability, e.g.,
 - Add new operations on the expression tree nodes without modifying the tree structure or implementation
 - Systematically reuse the expression tree processing app code in diverse runtime platforms, e.g.,
 - The app code is reused in both Android GUI & command-line platforms

	Run	n: 🗾	expression_tree	<	ΰ	t —
¥ 2: Favorites	■ * "D:\Douglas Schmid" >-5 * (3 + 4) -35				id	
	►	<u>4</u> : Run	= <u>0</u> : Messages	 ≄ <u>9</u> : Git	🔼 Terminal	A CMake
۳	-8	4 spaces	C++: expressio	on_tree Deb	ug 🗜 master	



Douglas C. Schmidt

Putting All the Pieces Together

Putting All the Pieces Together

 The expression tree processing app is a realistic case study of how to apply GoF patterns.

Design Problem	Pattern
Non-extensible & error-prone designs	Composite
Minimizing impact of variability	Bridge
Inflexible expression input processing	Interpreter
Inflexible interpreter output	Builder
Scattered request implementations	Command
Inflexible creation of variabilities	Factory Method
Inflexible expression tree traversal	Iterator
Obtrusive behavior changes	Strategy
Non-extensible tree operations	Visitor
Incorrect user request ordering	State
Non-extensible operating modes	Template Method
Minimizing global variable liabilities	Singleton

Putting All the Pieces Together

- The expression tree processing app is a realistic case study of how to apply GoF patterns.
 - All the case study code is written in C++ (~6K LOC & ~60 classes).

CPlusPlus/expression-tree at mas × +		- 🗆 X
\leftarrow \rightarrow C \triangle (a) github.com/douglascraigs \square Q \Rightarrow Q	💌 🛻 н 🗾 🚥	Ga 🛪 🚱 🗄
👖 Apps 🚥 🖪 🔇		
Branch: master + CPlusPlus / expression-tree /	Create new file Upload files Find	d file History
douglascraigschmidt updates	Latest commit 51438b	os on May 20
Component_Node.cpp	updates	last month
Component_Node.h	updates	last month
Composite_Add_Node.cpp	updates	last month
Composite_Add_Node.h	updates	last month
Composite_Binary_Node.cpp	updates	last month
Composite_Binary_Node.h	updates	last month
Composite_Divide_Node.cpp	updates	last month
Composite_Divide_Node.h	updates	last month
Composite_Multiply_Node.cpp	updates	last month
Composite_Multiply_Node.h	updates	last month
Composite_Negate_Node.cpp	updates	last month
Composite_Negate_Node.h	updates	last month
Composite_Subtract_Node.cpp	updates	last month
Composite_Subtract_Node.h	updates	last month
Composite_Unary_Node.cpp	updates	last month
Composite_Unary_Node.h	updates	last month
Evaluation_Visitor.cpp	updates	last month
Evaluation_Visitor.h	updates	last month
Event_Handler.cpp	updates	last month
Event_Handler.h	updates	last month
Expression_Tree.cpp	updates	last month
Expression_Tree.h	updates	last month
Expression_Tree_Command.cpp	updates	last month
Expression_Tree_Command.h	updates	last month
Expression_Tree_Command_Factory.cpp	updates	last month
Expression_Tree_Command_Factory.h	updates	last month
Expression_Tree_Command_Factory_Impl.cpp	updates	last month
Expression_Tree_Command_Factory_Impl.h	updates	last month

See github.com/douglascraigschmidt/CPlusPlus/tree/master/expression-tree

Putting All the Pieces Together

- The expression tree processing app is a realistic case study of how to apply GoF patterns.
 - All the case study code is written in C++.
 - There are command-line & Android GUI-based versions.

	Run	: 📃	expression_tree >	0	3	¢ —
		1	"D:\Do	uglas	s Schr	nid
/orites	-	→ IP	>-5 *	(3 +	4)	
¥ 2: Fav	*	**	-35			
	•	<u>4</u> : Run	Ξ <u>0</u> : Messages	 ≄ <u>9</u> : Git	E Terminal	A CMake
WT F	-8	4 space	s C++: expressio	n_tree Deb	ug 🗜 maste	r 🍙 💆

See github.com/douglascraigschmidt/CPlusPlus/tree/master/expression-tree