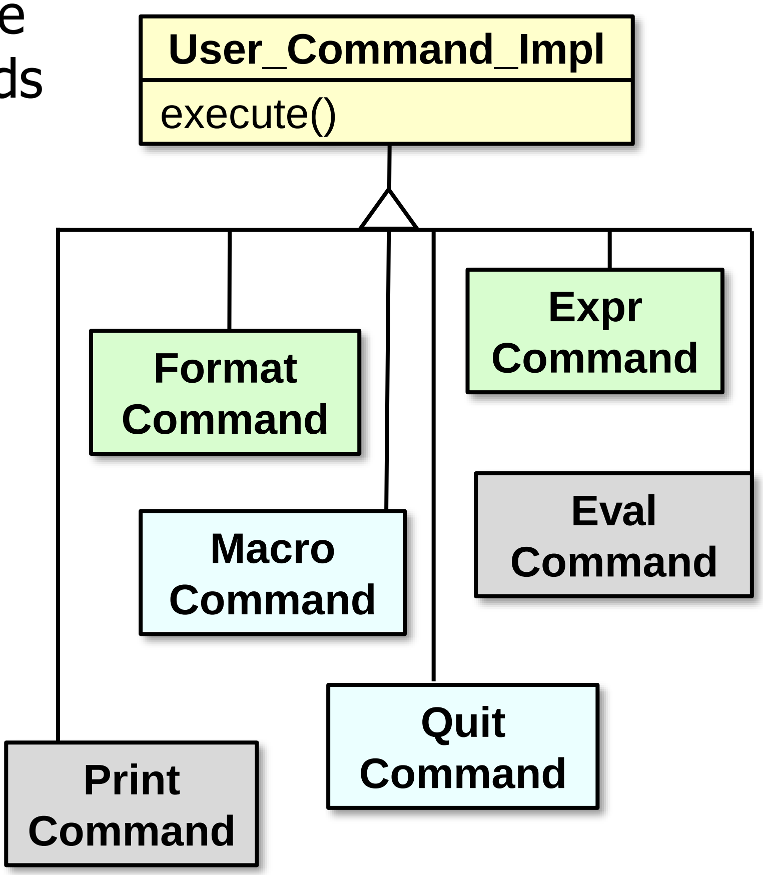
The Command Pattern

Motivating Example

Douglas C. Schmidt

Learning Objectives in This Lesson

- Recognize how the *Command* pattern can be applied to perform user-requested commands consistently & extensibly in the expression tree processing app.

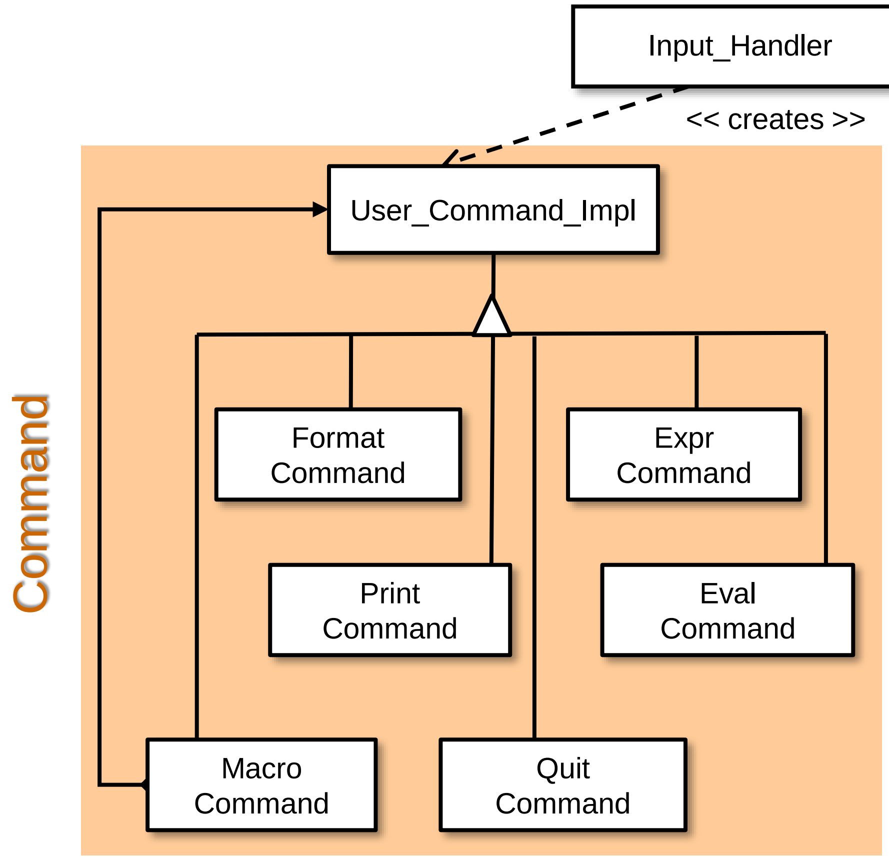


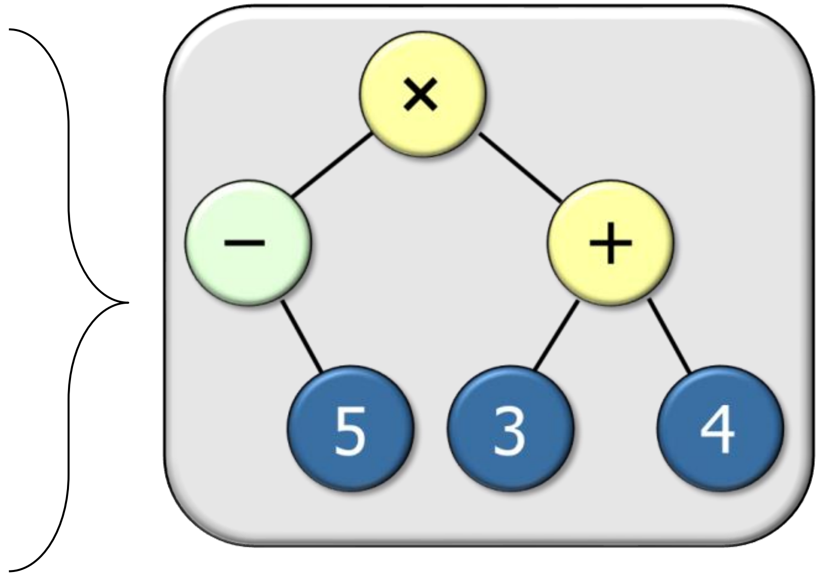
Douglas C. Schmidt

Motivating the Need for the Command Pattern in the Expression Tree App

A Pattern for Objectifying User Requests

***Purpose**: Define objectified actions that enable users to perform command requests consistently & extensibly in the expression tree processing app.*

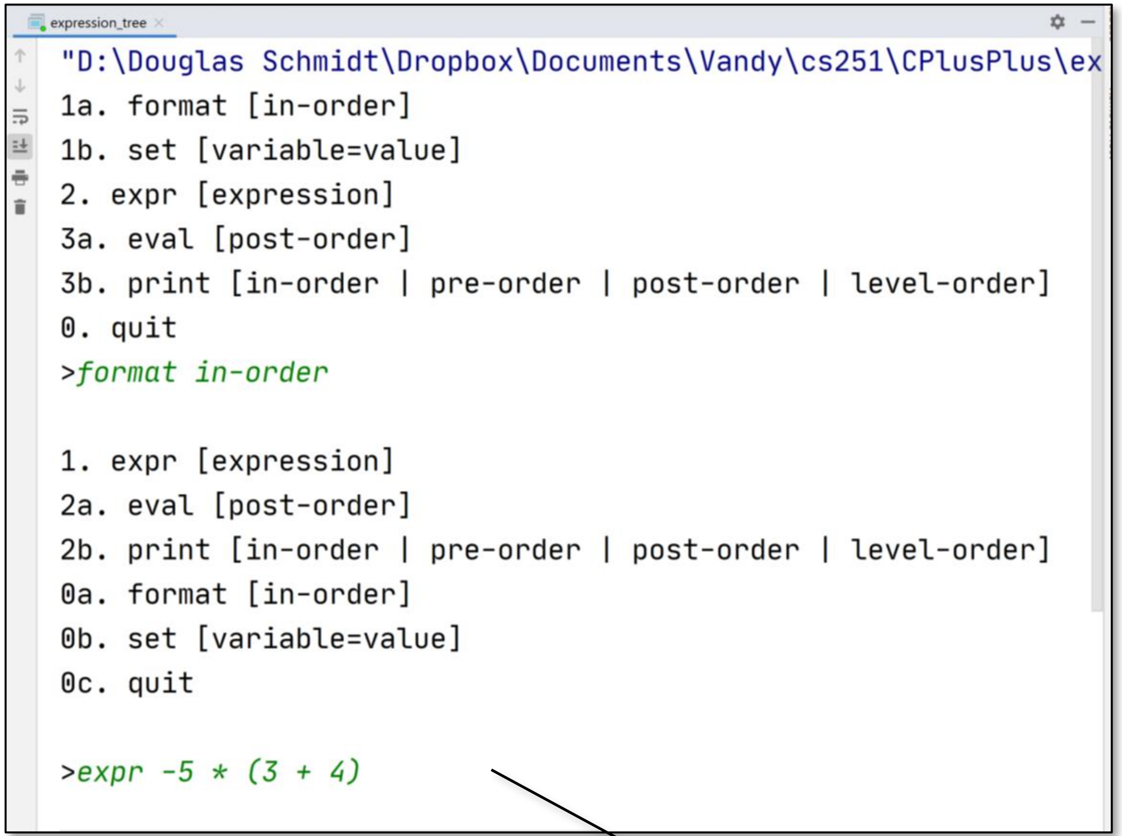




Command provides a uniform means to process all user-requested commands.

Context: OO Expression Tree Processing App

- Verbose mode supports user command execution

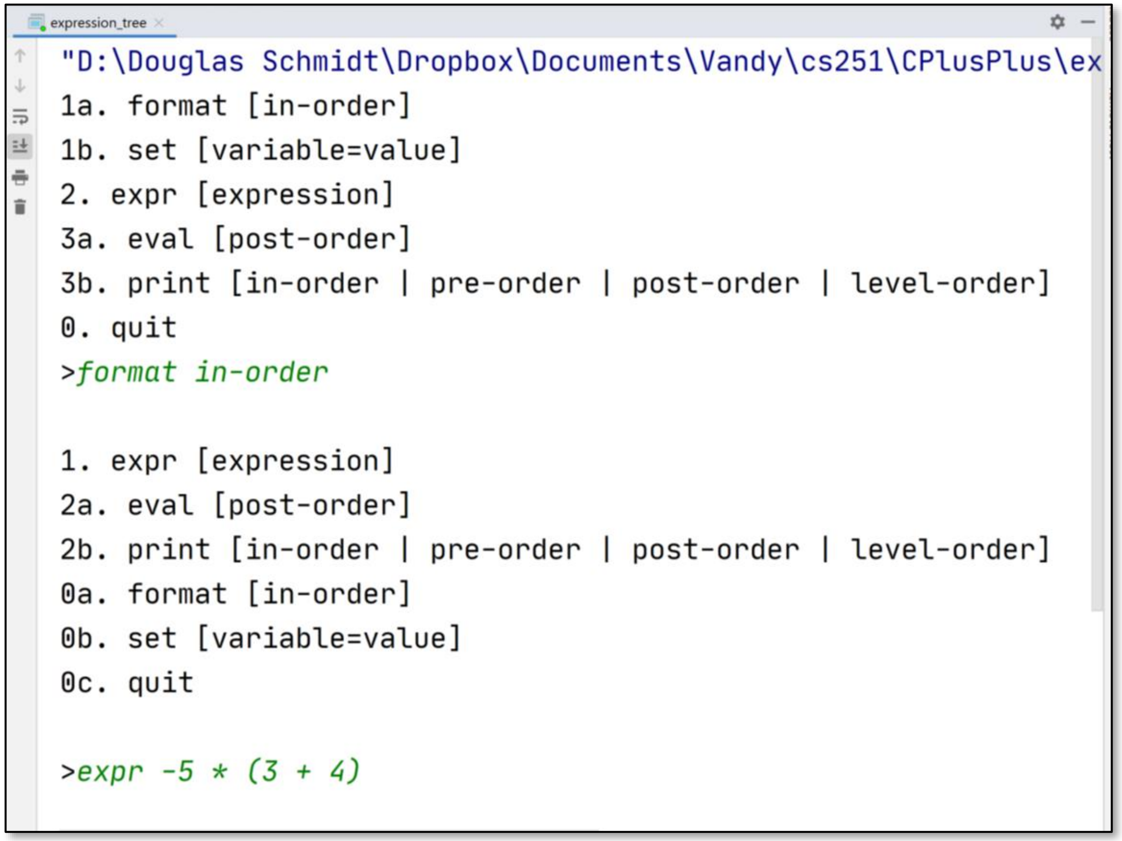


```
"D:\Douglas Schmidt\Dropbox\Documents\Vandy\cs251\CPlusPlus\ex
1a. format [in-order]
1b. set [variable=value]
2. expr [expression]
3a. eval [post-order]
3b. print [in-order | pre-order | post-order | level-order]
0. quit
>format in-order

1. expr [expression]
2a. eval [post-order]
2b. print [in-order | pre-order | post-order | level-order]
0a. format [in-order]
0b. set [variable=value]
0c. quit

>expr -5 * (3 + 4)
```

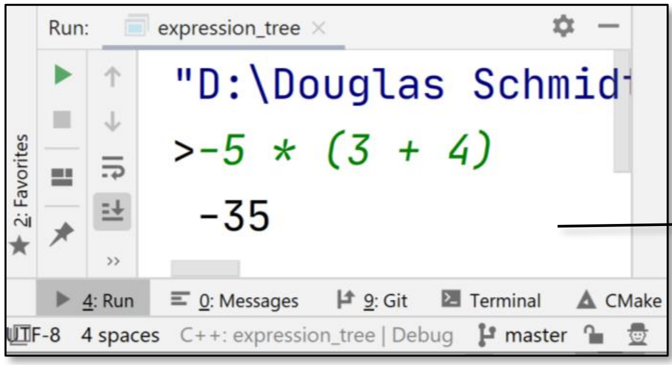
Verbose mode

Context: OO Expression Tree Processing App

- Succinct mode supports macro commands



```
"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]
0a. format [in-order]
0b. set [variable=value]
0c. quit

>expr -5 * (3 + 4)
```

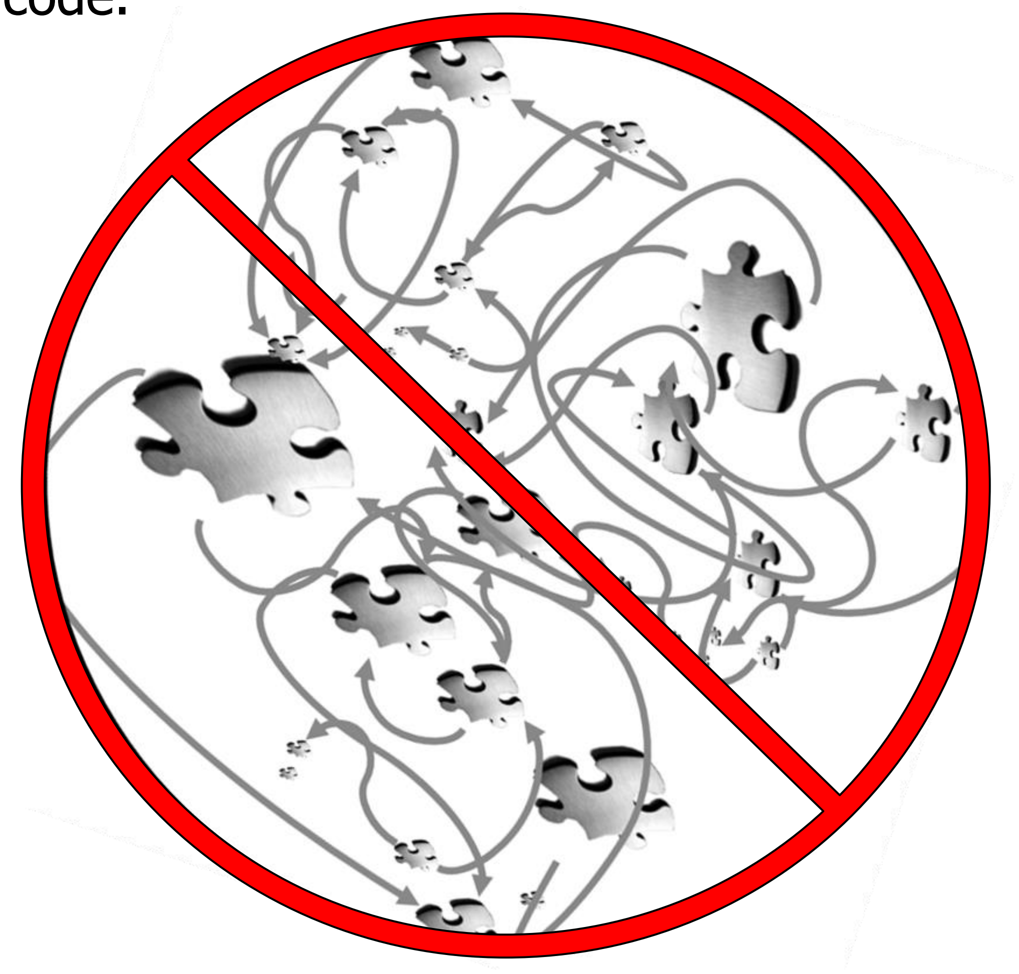


```
"D:\Douglas Schmidt
>-5 * (3 + 4)
 -35
```

Succinct mode

Problem: Scattered/Fixed User Request Implementations

- It's hard to maintain implementations of user-requested commands that are scattered throughout the source code.



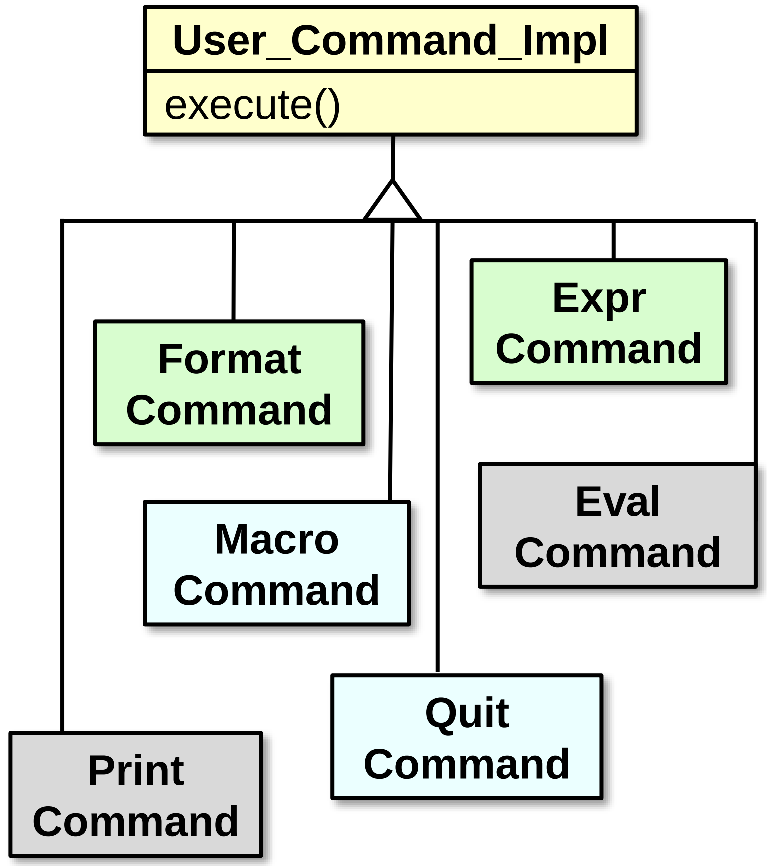
Problem: Scattered/Fixed User Request Implementations

- Hard-coding the program to handle only a fixed set of user commands impedes the evolution that's needed to support new requirements.

Operation
format
expr
set
print
eval
quit

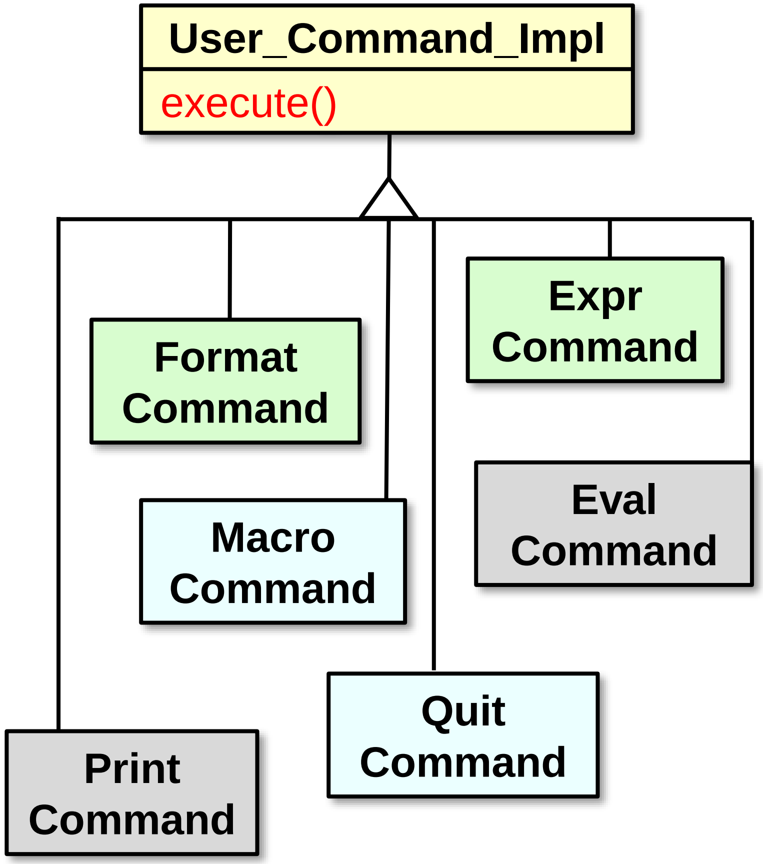
Solution: Encapsulate User Requests as Commands

- Create a hierarchy of `User_Command_Impl` derived classes



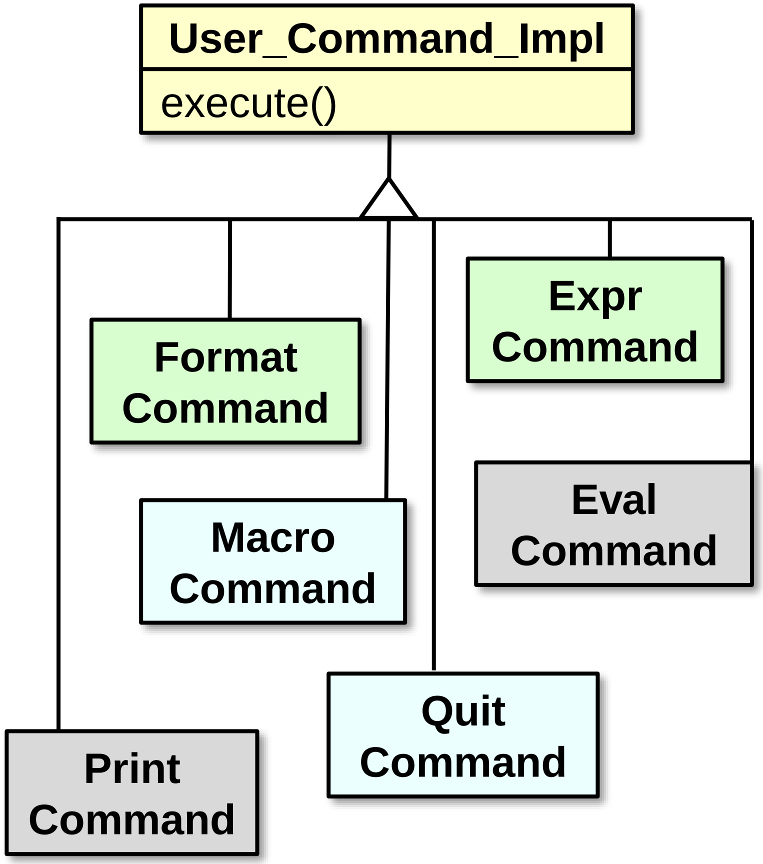
Solution: Encapsulate User Requests as Commands

- Create a hierarchy of **`User_Command_Impl`** derived classes, each containing:
  - A command method (**`execute()`**)



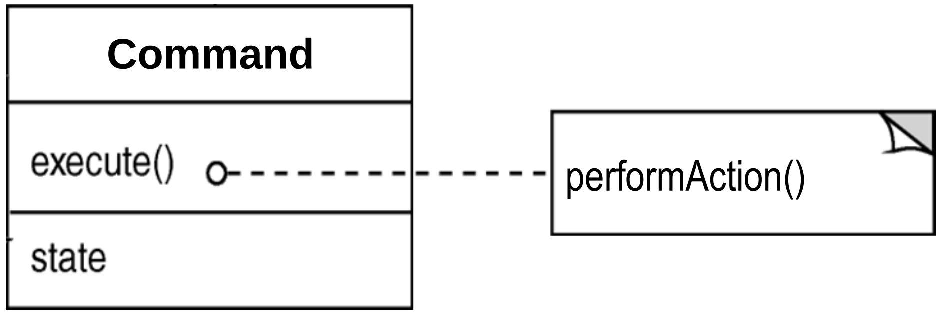
Solution: Encapsulate User Requests as Commands

- Create a hierarchy of `User_Command_Impl` derived classes, each containing:
  - A command method (`execute()`)
  - The state needed by the command



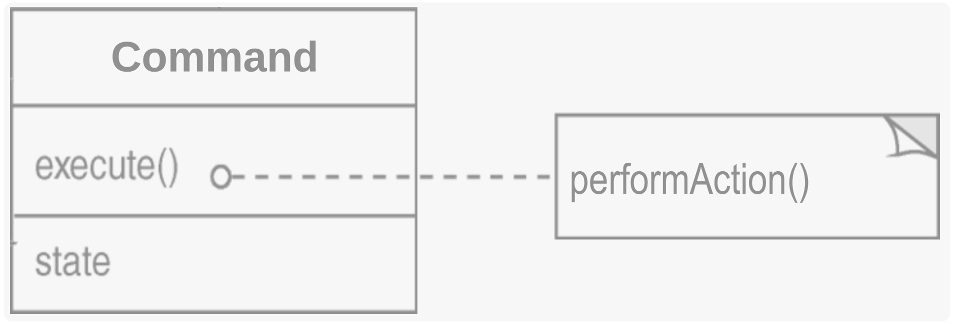
Solution: Encapsulate User Requests as Commands

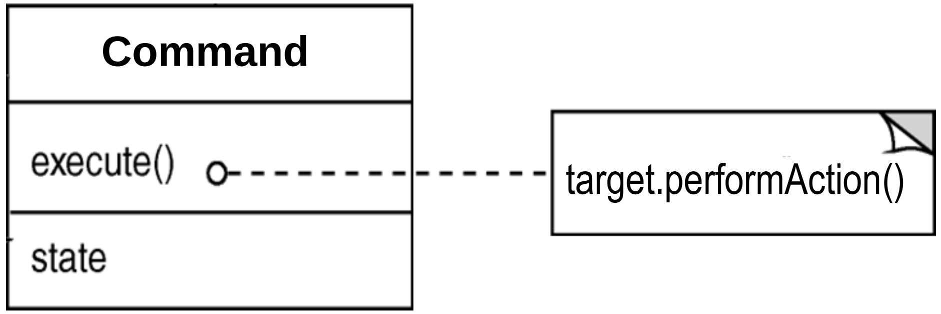
- A Command object may:
  - Implement the command itself



Solution: Encapsulate User Requests as Commands

- A Command object may:
  - Implement the operation itself
  - Or forward the command's implementation to other object(s)





The expression tree processing app applies this variant of the *Command* pattern

User_Command_Impl Class Overview

- Defines an abstract base class that performs a user-requested command on an expression tree when it's executed

Class methods

```
void execute()
void print_valid_commands()
```

User_Command_Impl Class Overview

- Defines an abstract base class that performs a user-requested command on an expression tree when it's executed

Class methods

These methods are defined by derived classes

```
void execute()
void print_valid_commands()
```

User_Command_Impl Class Overview

- Defines an abstract base class that performs a user-requested command on an expression tree when it's executed

Class methods

```
void execute()
void print_valid_commands()
```

- **Commonality**: provides a common API for expression tree commands
- **Variability**: derived classes of `User_Command_Impl` can vary depending on the commands requested by user input