The Command Pattern

Other Considerations

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.

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

• Know how to implement the *Command* pattern in C++.

• Be aware of other considerations when applying the *Command* pattern.
Douglas C. Schmidt

Other Considerations of the Command Pattern
Consequences

+ Abstracts the executor of a service
  • Makes programs more modular & flexible
Consequences

+ Abstracts the executor of a service

- Makes programs more modular & flexible, e.g.,
- Can bundle state & behavior into an object
Consequences

+ Abstracts the executor of a service
  - Makes programs more modular & flexible, e.g.,
    - Can bundle state & behavior into an object
  - Can forward behavior to other objects

See upcoming lesson on the *State* pattern for an example of forwarding.
Consequences

+ Abstracts the executor of a service
  
  • Makes programs more modular & flexible, e.g.,
    
    • Can bundle state & behavior into an object
    
    • Can forward behavior to other objects
    
    • Can extend behavior via derived classing
Consequences

+ Abstracts the executor of a service
  • Makes programs more modular & flexible, e.g.,
    • Can bundle state & behavior into an object
    • Can forward behavior to other objects
    • Can extend behavior via derived classing
    • Can pass a command object as a parameter

```c
void handle_input() {
    ...
    User_Command command = make_command(input);
    execute_command(command);
}
```

The `handle_input()` method in `Input_Handler` plays the role of “invoker.”
**Consequences**

+ Abstracts the executor of a service
  
  • Makes programs more modular & flexible, e.g.,
    
    • Can bundle state & behavior into an object
    
    • Can forward behavior to other objects
    
    • Can extend behavior via derived classing
    
    • Can pass a command object as a parameter

```java
void handle_input() {
    ... 
    User_Command command = make_command(input);
    Call a hook (factory) method to make a command based on user input

    execute_command(command);
```

See the next lesson on “*The Factory Method Pattern*” for *User_Command.Factory*. 

Consequences

+ Abstracts the executor of a service
  • Makes programs more modular & flexible, e.g.,
    • Can bundle state & behavior into an object
    • Can forward behavior to other objects
    • Can extend behavior via derived classing
  • Can pass a command object as a parameter

```cpp
void handle_input() {
    ...
    User_Command command = make_command(input);

    execute_command(command);

    Call a hook method & pass a command to execute
}
```

See upcoming lesson on “The Template Method Pattern”
Consequences

+ Composition yields macro commands

```
formatCommand().execute();
exprCommand().execute();
evalCommand().execute();
```
Consequences
+ Supports arbitrary-level undo-redo

**Undo:**
- `unexecute()`

**Redo:**
- `execute()`

Case study doesn’t use `unexecute()`, but it’s a common Command feature.
Consequences

– Might result in lots of trivial command derived classes
Consequences

- Excessive memory may be needed to support undo/redo operations

**Undo:**
- unexecute()
Implementation considerations

• Copying a command before putting it on a history list

**Undo:**
unexecute()  
past  
future

**Redo:**
execute()  
past  
future
Implementation considerations

- Avoiding error accumulation during undo/redo
Implementation considerations

- Supporting transactions

**Undo:**
- `unexecute()`

**Redo:**
- `execute()`

past \rightarrow future

past \rightarrow future
Known uses

- InterViews Actions
- MacApp, Unidraw Commands
- JDK’s UndoableEdit, AccessibleAction
- GNU Emacs
- Microsoft Office tools
- Java Runnable interface

See docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html
Known uses
- InterViews Actions
- MacApp, Unidraw Commands
- JDK’s UndoableEdit, AccessibleAction
- GNU Emacs
- Microsoft Office tools
- Java Runnable interface

Runnable can also be used to implement the Command Processor pattern

Packages a piece of application functionality—as well as its parameterization in an object—to make it usable in another context

See [www.dre.vanderbilt.edu/~schmidt/CommandProcessor.pdf](http://www.dre.vanderbilt.edu/~schmidt/CommandProcessor.pdf)
Summary of the Command Pattern

- *Command* ensures users interact with the expression tree processing app in a consistent & extensible manner.

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