The Command Pattern
Structure & Functionality

Douglas C. Schmidt
Learning Objectives in This Lesson

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

- Understand the structure & functionality of the *Command* pattern.
Structure & Functionality of the Command Pattern
Intent

• Encapsulate the request for a service as an object

See [en.wikipedia.org/wiki/Command_pattern](en.wikipedia.org/wiki/Command_pattern)
Applicability

- Want to parameterize objects with an action to perform

```
User_Command_Impl
  execute()
```

```
Format Command
Macro Command
Print Command
```

```
Expr Command
Eval Command
Quit Command
```
**Applicability**

- Want to parameterize objects with an action to perform
- Want to specify, queue, & execute requests at different times
Applicability

- Want to parameterize objects with an action to perform
- Want to specify, queue, & execute requests at different times
- Want to support multilevel undo/redo

We need to add an unexecute() method here.
Structure & participants

Command

GoF Object Behavioral

Input_Handler

Client

Invoker

Command

execute()

Target

action()

ConcreteCommand

execute()

state

target

target.action()
Command

GoF Object Behavioral

Structure & participants

- **Client**
- **Invoker**
- **Command**
  - `execute()`
- **Target**
  - `action()`
- **ConcreteCommand**
  - `execute()`
  - `state`

Command

Structure & participants

See upcoming lesson on the *State* pattern
Structure & participants

- **Client**
- **Invoker**
- **Command**
  - `execute()`
- **Target**
  - `action()`
- **ConcreteCommand**
  - `execute()`
  - `state`

The UI connects to the Client and Target, and the Client calls `target.action()`.
The **Client** & **Invoker** objects may be the same or different.