Applying Command Design Pattern

A Glimpse at Commander

S. Ducasse, L. Fabresse, G. Polito, and P. Tesone

http://www.pharo.org
Goals

- Command Design Pattern in Action
- Glimpse at Commander: a command framework
Commander: a Command framework

Commander is a little framework for commands using decorators

- Can produce a toolbar or menus
- UI is optional

(EgAddContactCommand new context: aPresenter) execute
Core commander
Add Contact

EgContactBookCommand <<= #EgAddContactCommand
package: 'EgContactBook'

CmAddContactCommand >> initialize
   super initialize.
   self
      basicName: 'New contact';
      basicDescription: 'Creates a new contact and add it to the contact book.'
Add Contact: Behavior

CmAddContactCommand >> execute
| contact |
contact := self contactBookPresenter newContact.
self hasSelectedContact
  ifTrue: [ self contactBook
    addContact: contact
    after: self selectedContact ]
  ifFalse: [ self contactBook addContact: contact ].
self contactBookPresenter updateView
Commander and its decorators
Commander and its decorators

CmCommand >> asSpecCommand
"Subclasses might override this method to define default icon and shortcut."
^ self decorateWith: SpCommand

StCommand >> asSpecCommand

| command |
command := super asSpecCommand
  iconProvider: self application;
  iconName: self class defaultIconName;
  yourself.
self class defaultShortcut
  ifNotNil: [:keyCombination | command shortcutKey: keyCombination ].
^ command
One Command

StCommand << StPlaygroundDoItCommand
package: 'NewTools-Playground'

StCommand >> execute
context doEvaluateAllAndGo
Conclusion

- Commands are first-class actions
- Adapted for manipulation of actions (undo, replay)