Command in Java

Command is behavioral design pattern that converts requests or simple operations into objects.

The conversion allows deferred or remote execution of commands, storing command history, etc.

Learn more about Command →

Navigation

Intro

Text editor commands and undo
commands
Command
CopyCommand
PasteCommand
CutCommand
CommandHistory
editor
Editor
Demo
OutputDemo

Complexity: ★☆☆

Popularity: ★★★

an alternative for callbacks to parameterizing UI elements with actions. It's also used for queueing tasks, tracking operations history, etc.

Here are some examples of Commands in core Java libraries:

- All implementations of `java.lang.Runnable`
- All implementations of `javax.swing.Action`

Identification: If you see a set of related classes that represent specific actions (such as "Copy", "Cut", "Send", "Print", etc.), this may be a Command pattern. These classes should implement the same interface/abstract class. The commands may implement the relevant actions on their own or delegate the work to separate objects—that will be the receivers. The last piece of the puzzle is to identify an invoker—search for a class that accepts the command objects in the parameters of its methods or constructor.

Text editor commands and undo

The text editor in this example creates new command objects each time a user interacts with it. After executing its actions, a command is pushed to the history stack.

Now, to perform the undo operation, the application takes the last executed command from the history and either performs an inverse action or restores the past state of the editor, saved by that command.

commands

commands/Command.java: Abstract base command

```
package refactoring_guru.command.example.commands;

import refactoring_guru.command.example.editor.Editor;

public abstract class Command {
    public Editor editor;
    private String backup;

    Command(Editor editor) {
        this.editor = editor;
    }
```

```
    }

    public void undo() {
        editor.textField.setText(backup);
    }

    public abstract boolean execute();
}
```

commands/CopyCommand.java: Copy selected text to clipboard

```
package refactoring_guru.command.example.commands;

import refactoring_guru.command.example.editor.Editor;

public class CopyCommand extends Command {

    public CopyCommand(Editor editor) {
        super(editor);
    }

    @Override
    public boolean execute() {
        editor.clipboard = editor.textField.getSelectedText();
        return false;
    }
}
```

commands/PasteCommand.java: Paste text from clipboard

```
package refactoring_guru.command.example.commands;

import refactoring_guru.command.example.editor.Editor;

public class PasteCommand extends Command {

    public PasteCommand(Editor editor) {
        super(editor);
    }

    @Override
    public boolean execute() {
        if (editor.clipboard == null || editor.clipboard.isEmpty()) return false;
```

```
        editor.textField.insert(editor.clipboard, editor.textField.getCaretPosition());
        return true;
    }
}
```

commands/CutCommand.java: Cut text to clipboard

```
package refactoring_guru.command.example.commands;

import refactoring_guru.command.example.editor.Editor;

public class CutCommand extends Command {

    public CutCommand(Editor editor) {
        super(editor);
    }

    @Override
    public boolean execute() {
        if (editor.textField.getSelectedText().isEmpty()) return false;

        backup();
        String source = editor.textField.getText();
        editor.clipboard = editor.textField.getSelectedText();
        editor.textField.setText(cutString(source));
        return true;
    }

    private String cutString(String source) {
        String start = source.substring(0, editor.textField.getSelectionStart());
        String end = source.substring(editor.textField.getSelectionEnd());
        return start + end;
    }
}
```

commands/CommandHistory.java: Command history

```
package refactoring_guru.command.example.commands;

import java.util.Stack;

public class CommandHistory {
```

```
    public void push(Command c) {
        history.push(c);
    }

    public Command pop() {
        return history.pop();
    }

    public boolean isEmpty() { return history.isEmpty(); }
}
```

editor

editor/Editor.java: GUI of text editor

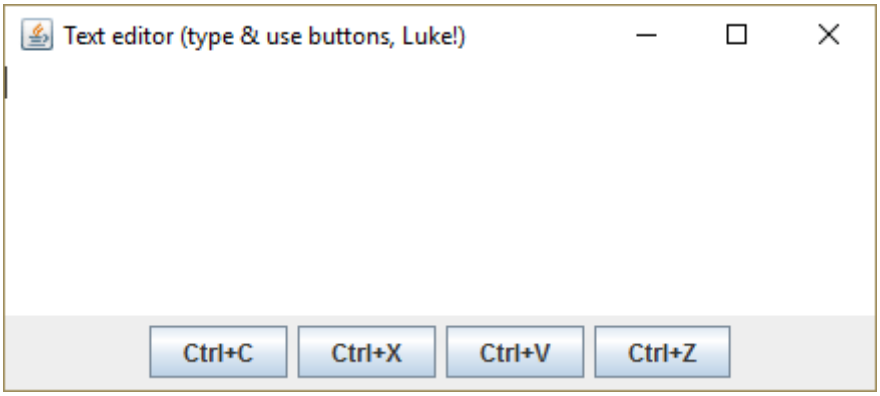
```
package refactoring_guru.command.example.editor;

import refactoring_guru.command.example.commands.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Editor {
    public JTextArea textField;
    public String clipboard;
    private CommandHistory history = new CommandHistory();

    public void init() {
        JFrame frame = new JFrame("Text editor (type & use buttons, Luke!)");
        JPanel content = new JPanel();
        frame.setContentPane(content);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        content.setLayout(new BoxLayout(content, BoxLayout.Y_AXIS));
        textField = new JTextArea();
        textField.setLineWrap(true);
        content.add(textField);
        JPanel buttons = new JPanel(new FlowLayout(FlowLayout.CENTER));
        JButton ctrlC = new JButton("Ctrl+C");
        JButton ctrlX = new JButton("Ctrl+X");
        JButton ctrlV = new JButton("Ctrl+V");
        JButton ctrlZ = new JButton("Ctrl+Z");
        Editor editor = this;
        ctrlC.addActionListener(new ActionListener() {
            @Override
```

```
            }
        });
        ctrlX.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                executeCommand(new CutCommand(editor));
            }
        });
        ctrlV.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                executeCommand(new PasteCommand(editor));
            }
        });
        ctrlZ.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                undo();
            }
        });
        buttons.add(ctrlC);
        buttons.add(ctrlX);
        buttons.add(ctrlV);
        buttons.add(ctrlZ);
        content.add(buttons);
        frame.setSize(450, 200);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    private void executeCommand(Command command) {
        if (command.execute()) {
            history.push(command);
        }
    }

    private void undo() {
        if (history.isEmpty()) return;

        Command command = history.pop();
        if (command != null) {
            command.undo();
        }
    }
}
```

Demo.java: Client code

```
import refactoring_guru.command.example.editor.Editor;

public class Demo {
    public static void main(String[] args) {
        Editor editor = new Editor();
        editor.init();
    }
}
```

OutputDemo.png: Execution result



READ NEXT

Iterator in Java →

RETURN

← Chain of Responsibility in Java