

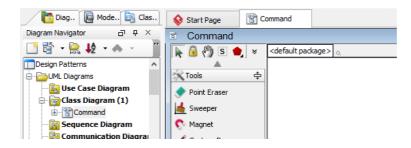
# **Command Pattern Tutorial**

Written Date : October 14, 2009

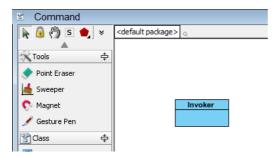
This tutorial is aimed to guide the definition and application of <u>Gang of Four (GoF)</u> command <u>design</u> <u>pattern</u>. By reading this tutorial, you will know how to develop a model for the command pattern, and how to apply it in practice.

### Modeling Design Pattern with Class Diagram

- 1. Create a new project *Design Patterns*.
- 2. Create a class diagram *Command*.



3. Select **Class** from diagram toolbar. Click on the diagram to create a class. Name it as *Invoker*.



4. Move the mouse cursor over the *Invoker* class, and drag out **Aggregation** > **Class** to create an associated class *Command*.

Invoker		Command
	$\diamond$ —	

5. Right-click on *Command*, and select **Model Element Properties > Abstract** to set it as abstract.

Command	ſ		
	Add	×	
	Open Specification	Enter	
	Stereotypes	•	
	Model Element Properties	•	Visibility 🕨
	Sub Diagrams	•	Abstract
	Create Parent	•	4

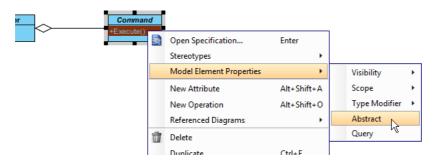
6. Right-click on *Command* class, and select **Add** > **Operation** from the popup menu.

r Comma	nd	[			
		Add	•	Attribute	Alt+Shift+A
		Open Specification	Enter	Attribute with Getter and Setter	
		Stereotypes		Operation	Alt+Shift+O
		Model Element Properties	•	Constructor	
		Sub Diagrams	•	Template Parameter	
		Create Parent	•		
	. e. e.				

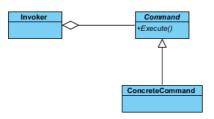
7. Name the operation *Execute()*.



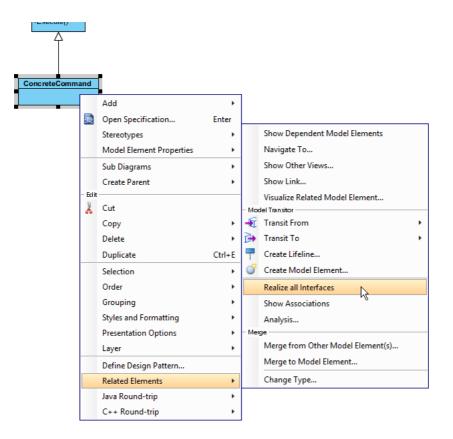
8. Right-click on *Execute*, and select **Model Element Properties > Abstract** to set it as abstract.



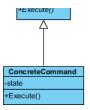
9. Move the mouse cursor over the *Command* class, and drag out **Generalization** > **Class** to create subclasses *ConcreteCommand*.



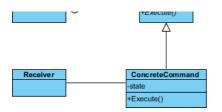
 We need make the concrete commands inherit operations from the command class. Right-click on *ConcreteCommand* and select **Related Elements** > **Realize all Interfaces** from the popup menu.



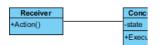
11. Right-click on the *ConcreteCommand* class, and select **Add** > **Attribute** from the popup menu. Enter *state* as attribute name.



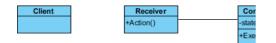
12. Move the mouse cursor over the *ConcreteCommand* class, and drag out **Association** > **Class** to create an associated class *Receiver*.



13. Right-click on the *Receiver* class, and select **Add** > **Operation** from the popup menu. Enter *Action* as operation name.



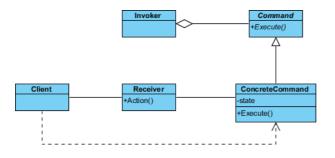
14. Create a *Client* class near the *Receiver* class.



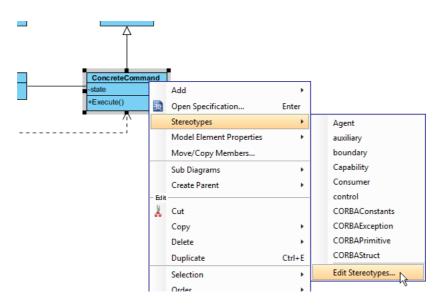
15. Move the mouse cursor over the *Client* class, and drag out **Association** > **Class** to create an associated class *Receiver*.



16. Move the mouse cursor over the *Client* class, and drag out **Dependency** > **Class** to create an associated class *ConcreteCommand*. Up to now, the diagram becomes:



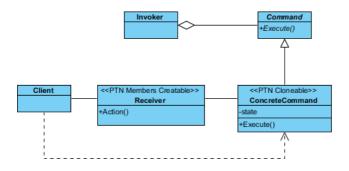
17. In practice, there may be multiple concrete handlers. To represent this, stereotypes the class *ConcreteCommand* as **PTN Cloneable**. Right-click on *ConcreteCommand* and select **Stereotypes > Stereotypes...** from the popup menu.



18. In the Stereotypes tab of the Class Specification dialog box, select PTN Cloneable and click
 > to assign it to ConcreteCommand class. Click OK to confirm.

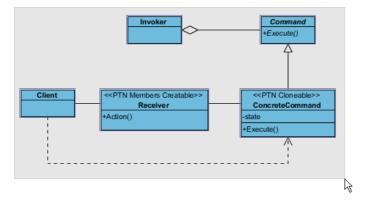
Class Specification	×
General         Attributes         Operations         Relations         Chart Relations         Template Parameters           Diagrams         Traceability         References         Project Management         Quality         Comments           Class Code Details         Java Annotations         Stereotypes         Tagged Values         Constraints           All:         Selected:         Selected:         Selected:         Selected:	* *
Image: Second Secon	
Reset OK Cancel Apply Help	

19. There may be multiple actions that the receiver can perform. To represent this, stereotype the class *Receiver* as **PTN Members Creatable**. Up to now, the diagram becomes:

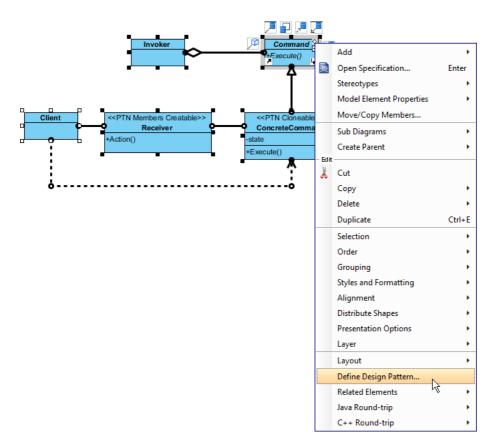


## **Defining Pattern**

1. Select all classes on the class diagram.



2. Right-click on the selection and select **Define Design Pattern...** from the popup menu.



3. In the **Define Design Pattern** dialog box, specify the pattern name *Command*. Keep the file name as is. Click **OK** to proceed.

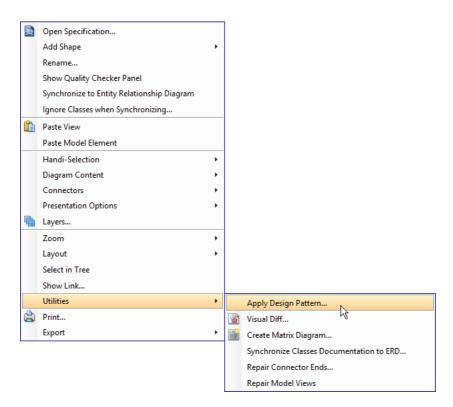
<b>\$</b>	Define Design Pattern ×
Name:	Command
File name:	Command.pat
Location	to workspace:
	to workspace:
Ŭ	C: \Users \John \Applications \Visual Paradigm 11. 1\bin \vpworkspace \vp_design_pattern_repo     v
Destination	: C:\Users\John\Applications\Visual Paradigm 11.1\bin\vpworkspace\vp_design_pattern_repo\Command.pat
	OK Cancel

### **Applying Design Pattern on Class Diagram**

In this section, we are going to apply the command pattern in modeling a document editor.

- 1. Create a new project *Document Editor*.
- 2. Create a class diagram Domain Model.

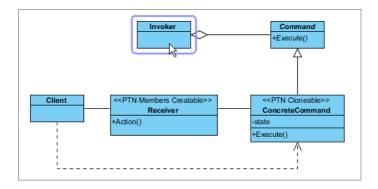
3. Right-click on the class diagram and select **Utilities** > **Apply Design Pattern...** from the popup menu.



4. In the **Design Pattern** dialog box, select *Command* from the list of patterns.

<b>\$</b>		Design Pattern
Patterns: Command		Invoker +Execute()
	Client	< <ptn creatable="" members="">&gt; Receiver +Action() +Action() -state +Execute()</ptn>
	Diagram Element <all></all>	~
	Auto Rename	Client
	Command	Client V Command V
	😝 Execute	Execute V
	ConcreteCommand	ConcreteCommand v +
	😝 Execute	Execute
	😑 state	state v
	Invoker	Invoker V
	Receiver	Receiver v +
	Action	Action
Add Remove		OK Cancel

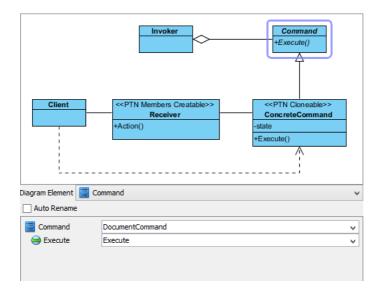
5. Select *Invoker* in overview.



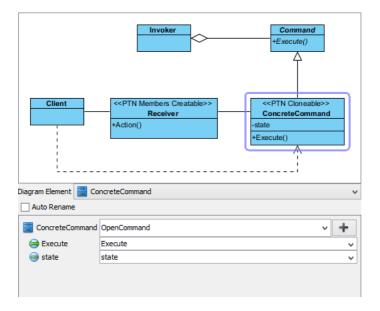
6. At the bottom pane, rename *Invoker* to *ToolbarButton*.

📑 Invoker	ToolbarButton	~

7. Select Command in overview. At the bottom pane, rename Command to DocumentCommand.



8. Select *ConcreteCommand* in overview. At the bottom pane, rename *ConcreteCommand* to *OpenCommand*.



9. We need 2 more concrete commands for closing and saving a document. Press on the + button and select **Clone...** from the popup menu.



10. Enter 2 to be the number of classes to clone.

	Input ×
?	Please enter the number of classes to clone. 2 OK Cancel

11. Rename ConcreteCommand2 to CloseCommand, ConcreteCommand3 to SaveCommand.

Client	<pre></pre> <pre>&lt;</pre>	Command +Execute()
Diagram Element 📑 Con	creteCommand	¥
ConcreteCommand	OpenCommand	v 🛨
😝 Execute	Execute	×
😑 state	state	¥
ConcreteCommand2	CloseCommand	¥
😂 Execute	Execute	×
😑 state	state	~
ConcreteCommand3	SaveCommand	~
😂 Execute	Execute	~
😑 state	state	~

12. Select *Receiver* in overview. At the bottom pane, rename *Receiver* to *Document*, and operation *Action* to *Load*.

Client	<ptn creatable="" members="">&gt; Receiver +Action()</ptn>	Command +Execute() < <ptn cloneable="">&gt; ConcreteCommand -state +Execute()</ptn>
Diagram Element  🗐 Re	eceiver	Ý
Auto Rename		
EReceiver	Document Load	* +

13. Create more operations for closing and saving documents. Click on the + button and select **New Operation...** from the popup menu.

teiver	Document	~	+				
ction	Load		1	New Attribute			
			1	lew	Operation		
					43		

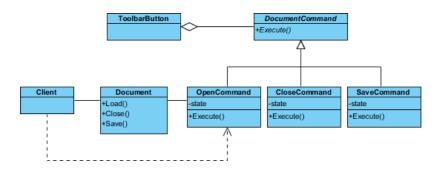
14. In the Operation Specification, enter Close as name. Click OK to confirm.

<b>\$</b>		Oper	ation Specifi	cation			×
Reference Stereotypes	s Tagge	Project Mar d Values	Relations	Quality Constraints	Comme Tracea	ability	ŕ
General	n Code Details Parameters		Java Annotations Exceptions	Preconditions	ate Paramete Postcond		2
Name:	Close						
Classifier:	📑 Receiver						
Return type:					¥	*	
Type modifier:	<unspecified></unspecified>					~	
Visibility:	public					~	
Scope:	instance					~	
Lower:							

15. Repeat steps 13 and 14 to create operation Save.

+	Receiver
~	😂 Action
~	<ul> <li>⇔ Action</li> <li>⇔ Close</li> <li>⇔ Save</li> </ul>
~	😝 Save
	😸 Save
	Jave Save

- 16. Click **OK** to apply the pattern to diagram.
- 17. Tidy up the diagram. Here is the result:



#### Resources

- 1. <u>Command.pat</u>
- 2. Design Patterns.vpp

#### **Related Links**

• Full set of UML tools and UML diagrams



(https://www.visual-paradigm.com/)

Visual Paradigm tutorials (https://www.visual-paradigm.com/tutorials/)