

# 10

# Command

## Diseño del Software

Grado en Ingeniería Informática del Software

2024-2025

# **Patrón de comportamiento de objetos**

*Encapsula una petición dentro de un objeto, permitiendo parametrizar a los clientes con distintas peticiones, guardarlas, registrarlas o implementar un mecanismo de deshacer/repetir.*

**También conocido  
como  
Action, Transaction**

# Motivación

A veces es necesario enviar peticiones a objetos sin saber nada ni de la operación concreta a ejecutar ni del receptor de la petición.

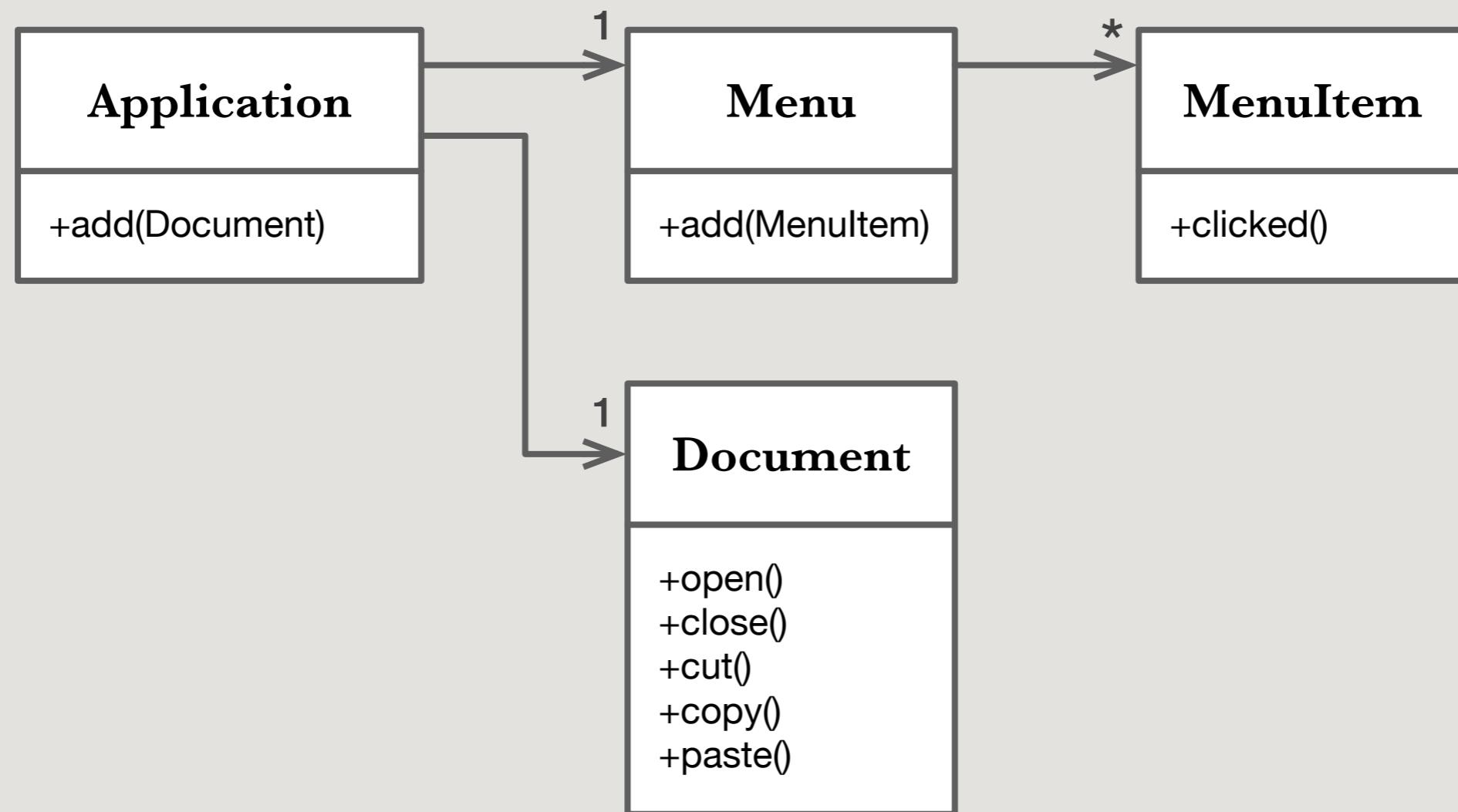
A veces es necesario enviar peticiones a objetos sin saber nada ni de la operación concreta a ejecutar ni del receptor de la petición.

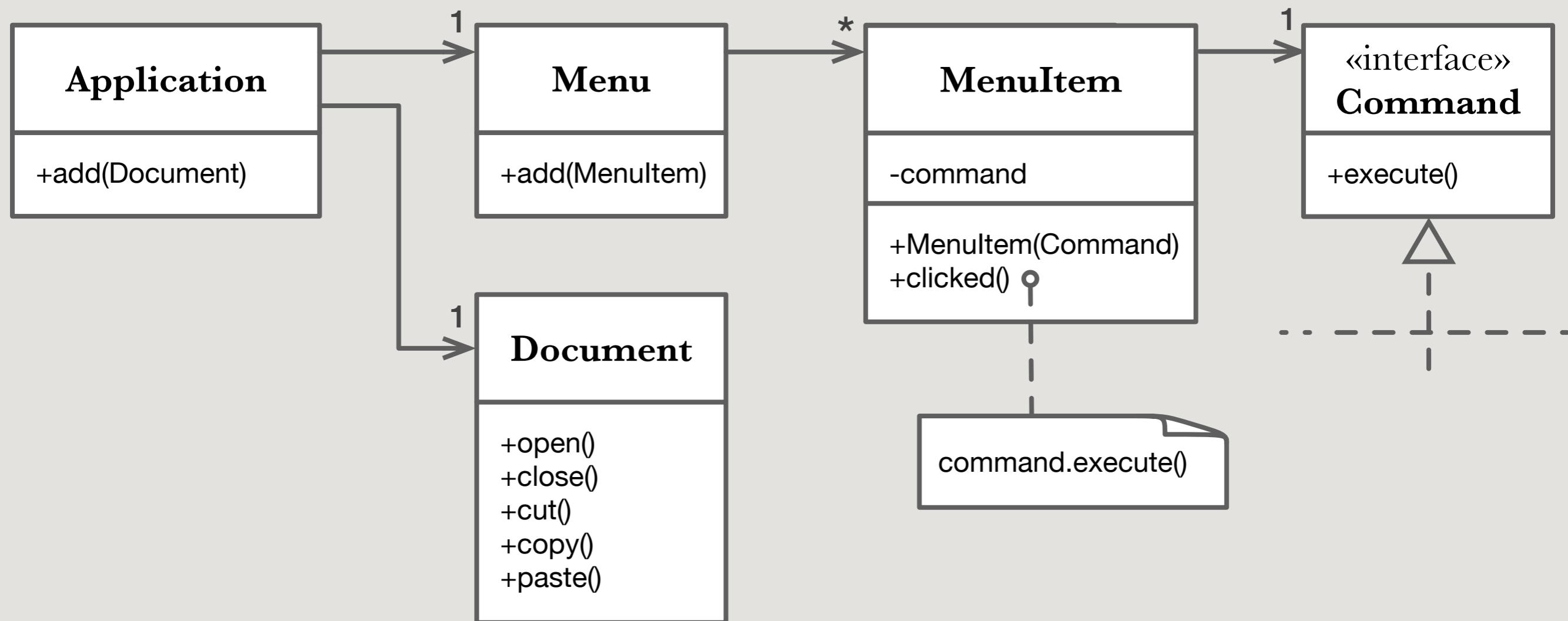
A veces es necesario enviar peticiones a objetos sin saber nada ni de la operación concreta a ejecutar ni del receptor de la petición.

Una biblioteca para interfaces de usuario tendrá objetos como botones y elementos de menú responsables de realizar alguna operación en respuesta a una entrada del usuario.

La biblioteca no puede implementar dichas operaciones directamente en el botón o el menú, porque solo las aplicaciones que usan la biblioteca saben qué hay que hacer y a qué operaciones de otros objetos hay que llamar.

Los diseñadores de la biblioteca no tienen forma de conocer al receptor de la petición ni las operaciones que la llevarán a cabo.



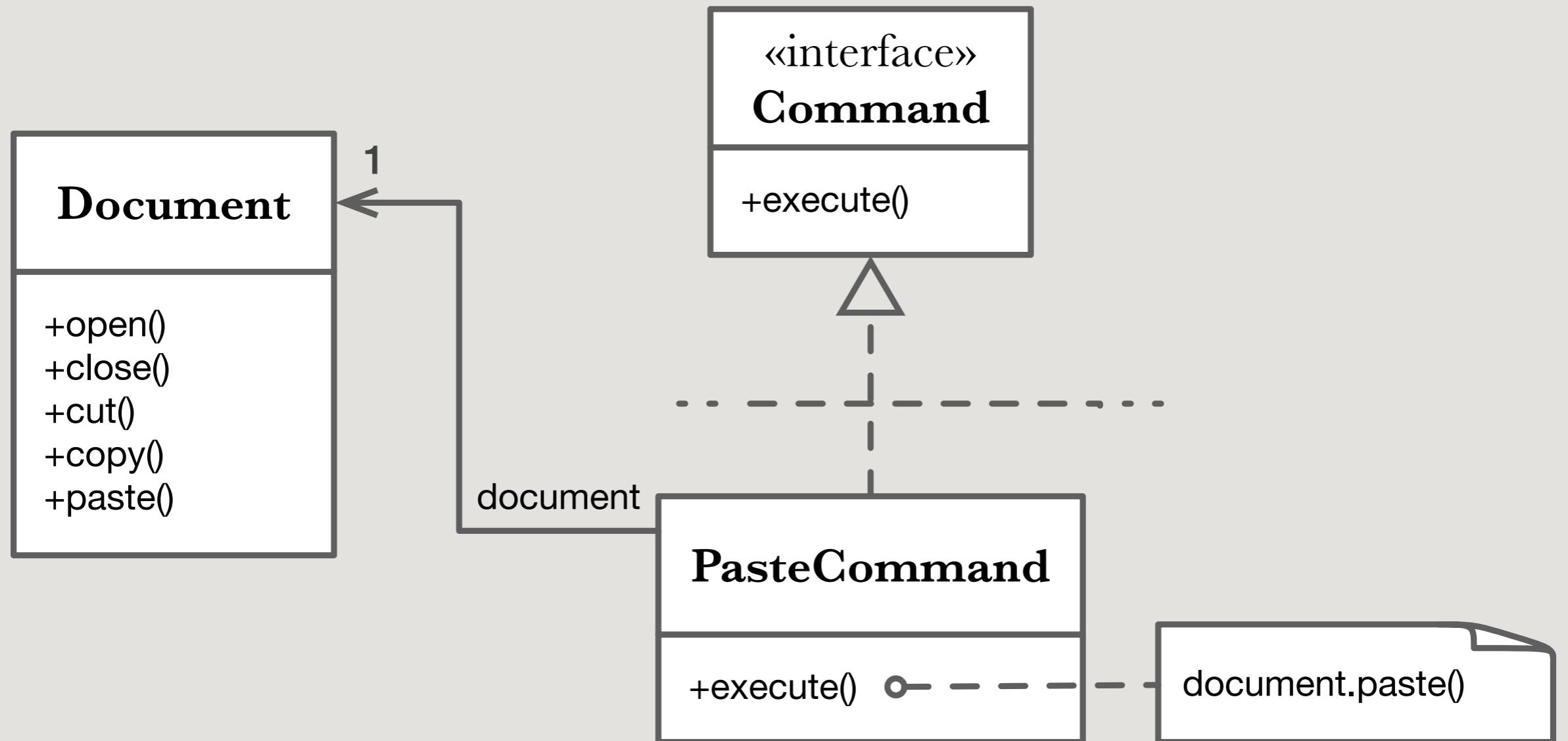


El patrón Command permite a los objetos de la biblioteca llamar a operaciones no especificadas de objetos desconocidos convirtiendo la propia petición en un objeto.

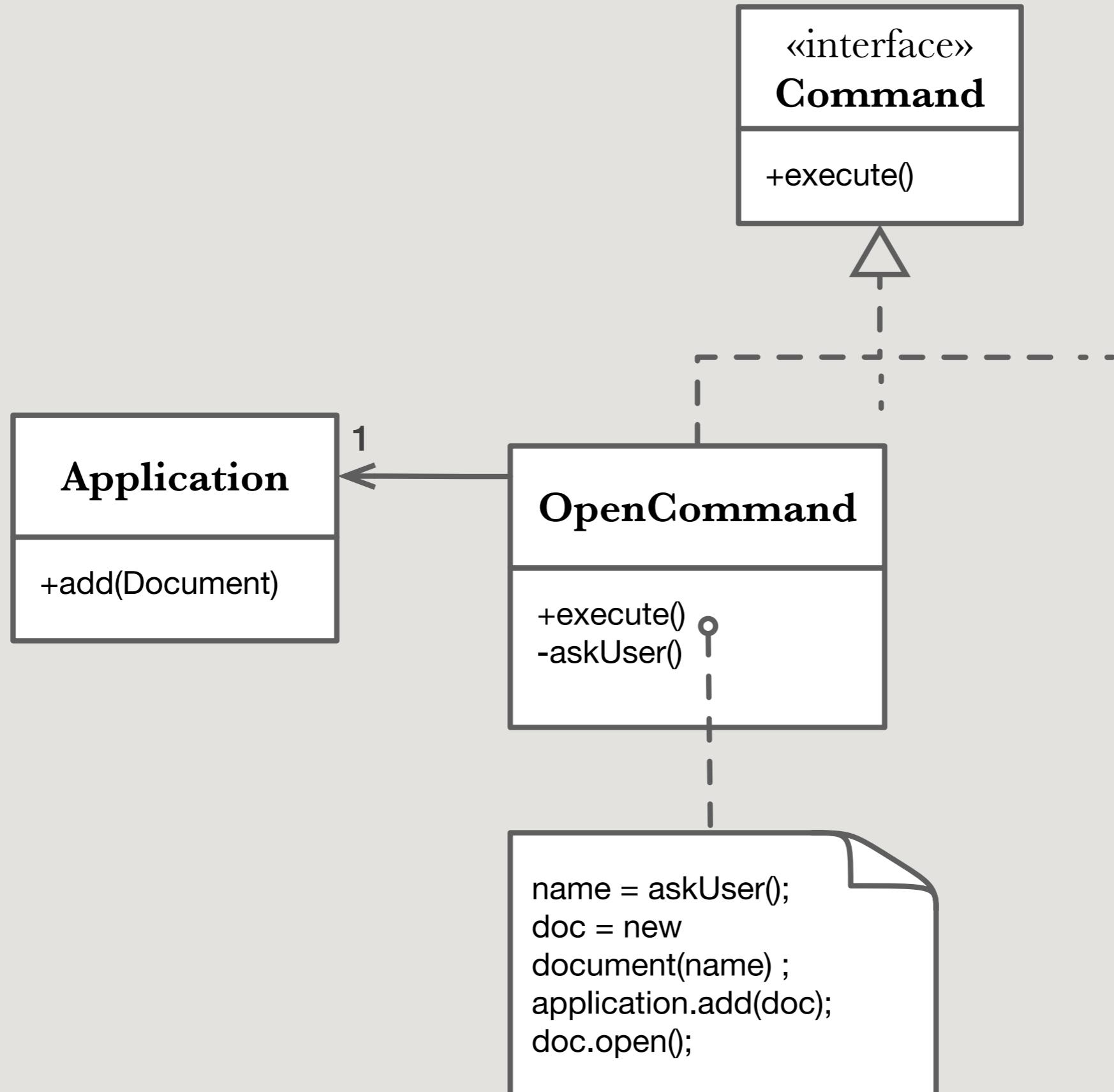
La cual puede guardarse y pasarse como parámetro como cualquier otro objeto.

Las clases ConcreteCommand especifican un par receptor-acción almacenando el receptor como un atributo e implementando execute para invocar la petición.

El receptor es quien tiene el conocimiento necesario para llevar a cabo la acción.



Pegar



Abrir

En resumen, lo que permite el patrón Command es desacoplar al objeto que invoca a la operación de aquel que tiene el conocimiento necesario para realizarla.

**Esto nos otorga muchísima flexibilidad.**

Podemos hacer, por ejemplo, que una aplicación proporcione tanto un elemento de menú como un botón para hacer una determinada acción.

Podemos cambiar dinámicamente los objetos Command.

Etcétera.

# Aplicabilidad

Úsese el patrón Command  
cuando se quiera

# Parametrizar objetos con una determinada acción a realizar.

Como los objetos Menultem anteriores.

Es lo que haríamos en un lenguaje procedimental con una función callback o, lo que es lo mismo, un puntero a función que se guarda en algún sitio para ser llamada más tarde.

Los commands son un sustituto orientado a objetos de los callbacks.

*en un momento posterior*



# Especificar, guardar y ejecutar la petición en distintos momentos.

Un objeto command puede tener un **ciclo de vida** independiente de la petición original.

«desacoplamiento temporal»

# Desacoplamiento temporal

El patrón desacopla no solo quién crea (normalmente, el cliente) y quién la llama (el «*invoker*»), o a este de qué se está ejecutando (el *command* en sí), sino sobre todo **cuándo** se ejecuta.

Es la esencia del patrón Command.

Es lo que lo hace útil para la ejecución diferida, mecanismos de deshacer/repetir, etcétera.

# **Permitir deshacer/repetir.**

En ese caso, execute deberá guardar el estado necesario para poder revertir los efectos de ejecutar la operación.

Y añadir una operación undo que revertirá los efectos de una llamada previa a execute. Las acciones realizadas se guardan en un historial.

# **Registrar los cambios para que se puedan volver a aplicar en caso de un fallo del sistema.**

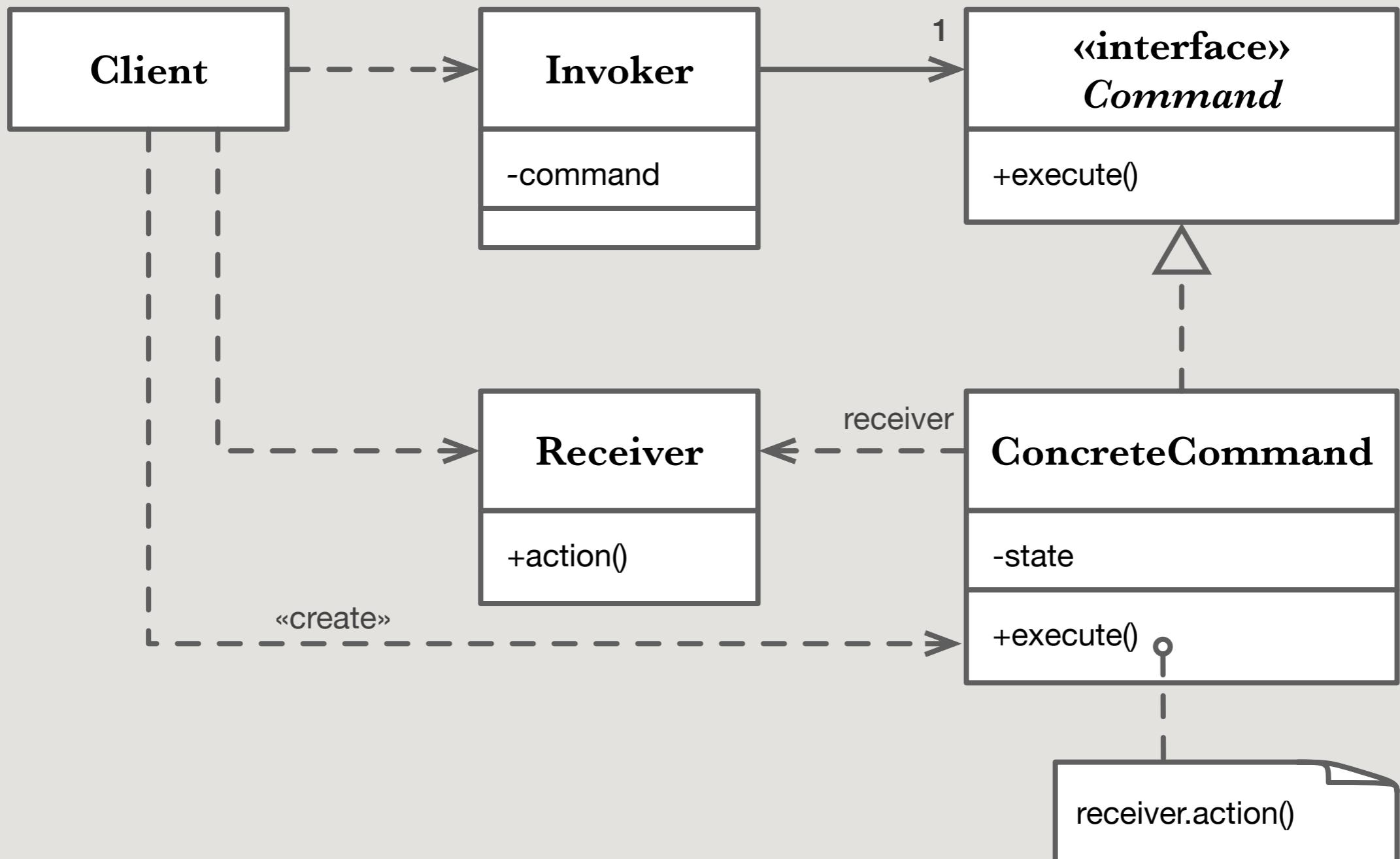
Añadiendo un par de operaciones **load** y **store** a la interfaz del Command para tener dicho registro de cambios persistente.

La recuperación de un fallo implica volver a cargar los comandos registrados desde el disco y volver a ejecutarlos con la operación de ejecución.

**Estructurar un sistema en torno a operaciones de alto nivel basadas en operaciones primitivas.**

El patrón Command ofrece una forma de modelar dichas transacciones.

# Estructura



La estructura del patrón Command

# Participantes

# Command

(Command)

**Define una interfaz para ejecutar una operación.**

# ConcreteCommand

(PasteCommand, OpenCommand)

**Asocia un objeto receptor con una acción.**

**Implementa execute llamando a las operaciones de dicho objeto receptor.**

# Client

(Application)

Crea un objeto **ConcreteCommand** y establece su receptor.

Lo guarda en el invoker.

# Invoker

(MenuItem)

Le pide al command que se ejecute.

# Receiver

(Document, Application)

**Sabe cómo llevar a cabo las operaciones asociadas con una petición.**

Puede ser cualquier clase.

# Colaboraciones



1

## **El cliente crea un objeto ConcreteCommand y especifica quién es su receptor.**

Normalmente, pasándoselo en el constructor en el momento de la creación, junto con cualquier información adicional necesaria para ejecutar la acción más tarde.

Dicho receptor será en quien delegue posteriormente la acción concreta para llevar a cabo su cometido cuando se llame a su método execute.

2

Un objeto Invoker guarda dicho objeto  
ConcreteCommand.



3

En algún momento posterior, dicho objeto Invoker se encargará de llamar a la operación execute del Command.

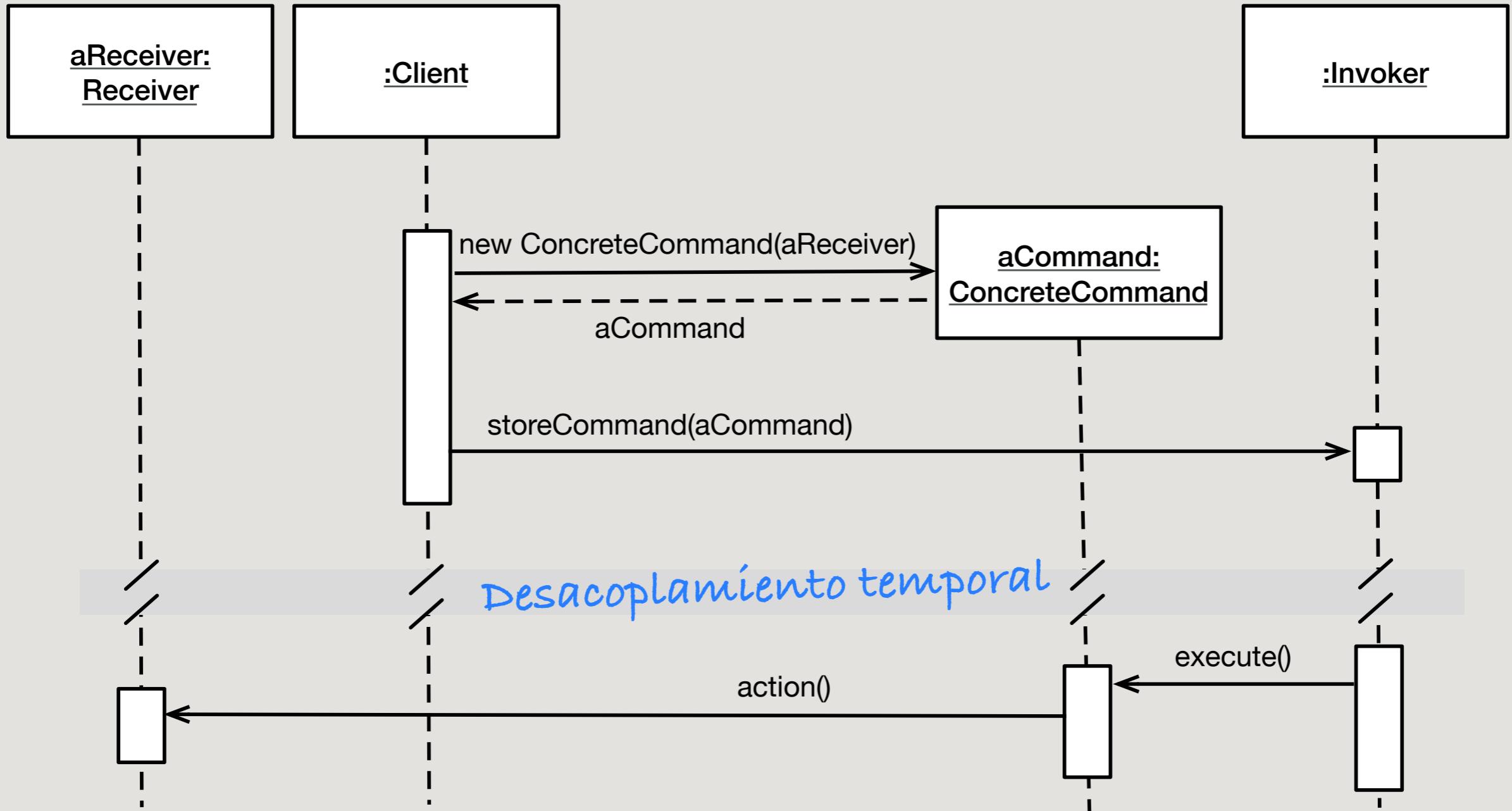
Quien previamente guardará el estado para luego poder deshacer la operación, en el caso de que se trate de operaciones que se puedan deshacer).



**4**

Por último, el **ConcreteCommand** se vale de las operaciones de su receptor para llevar a cabo la acción.

El siguiente diagrama muestra las interacciones entre estos objetos.



Ilustra cómo el patrón Command desacopla el objeto invoker del receptor y la petición que lleva a cabo (el command concreto), así como el objeto que sabe qué acción hay que realizar (el cliente), del que sabe cuándo hay que ejecutarla (el invoker).

# Consecuencias

**Desacopla el objeto que llama a la operación del que sabe cómo llevarla a cabo.**

Además, separa la creación de la acción concreta (qué operación debe realizarse), del objeto que sabe cuándo debe ejecutarse.

# Son objetos de primera clase.

Lo que significa que pueden pasarse como parámetros, guardarse en una variable y tratarse polimórficamente, como cualquier otro objeto.

## Se pueden ensamblar.

El patrón Command se utiliza a menudo junto con el patrón Composite para crear acciones compuestas, como **macros** o **transacciones**.

# Resulta sencillo añadir nuevas acciones.

Al no tener que cambiar las clases existentes (cumpliendo así el principio de abierto-cerrado).

# Implementación

**¿Cómo de inteligente  
debería ser?**

Un command puede hacer desde casi nada a todo el trabajo.

En un extremo, no es más que una mera encapsulación de la petición, asociando el receptor y la operación a ejecutar.



```
public class ConcreteCommand {  
  
    private Receiver aReceiver;  
  
    public ConcreteCommand(Receiver aReceiver) {  
        this.aReceiver = aReceiver;  
    }  
  
    public void execute() {  
        aReceiver.action();  
    }  
}
```

En el otro, podría implementarlo él todo sin delegar en ningún otro objeto (no habría receptor).

Esto es útil cuando queremos definir acciones que sean independientes de las clases existentes o cuando no existe un receptor adecuado.

DxO PhotoLab – Command – Gilbo.tif (M)

Compare 1:1 62 % Reset Presets

HISTOGRAM

RGB R G B L

219, 220, 220

MOVE/ZOOM

Move/Zoom

Advanced History

Applied Default Preset

Clear History

PRESET EDITOR

Command

Gilbo.tif

1/1 images

Search for corrections

Exposure Correction Manual Exposure 0,00

DxO Smart Lighting

Uniform Spot Weighted

Mode Slight Intensity 25

Selective Tone

Highlights 0 Midtones 0 Shadows 0 Blacks 0

DxO ClearView Plus

Intensity 50

Contrast

Contrast 0 Microcontrast 0

Tone Curve

Curve Master Reset Reset All

Vignetting

Correction Manual Intensity 100

Nik Collection Export

The screenshot displays the DxO PhotoLab software interface with the following details:

- Title Bar:** DxO PhotoLab – Command – Gilbo.tif (M)
- Top Bar:** Compare, 1:1, 62 %, Reset, Presets
- Histogram Panel:** Shows the color channel distribution (RGB, R, G, B, L) with a total count of 219, 220, 220.
- Move/Zoom Panel:** Includes a preview window showing a different crop of the image.
- History Panel:** Advanced History, Applied Default Preset, Clear History.
- Preset Editor Panel:** Command, Gilbo.tif.
- Image Preview:** A large central view of a rugged mountain landscape with a prominent rocky foreground and a winding river.
- Right Side Panels (Adjustments):**
  - Exposure: Correction Manual, Exposure 0,00
  - DxO Smart Lighting: Uniform, Spot Weighted, Mode Slight, Intensity 25
  - Selective Tone: Highlights 0, Midtones 0, Shadows 0, Blacks 0
  - DxO ClearView Plus: Intensity 50
  - Contrast: Contrast 0, Microcontrast 0
  - Tone Curve: Curve Master, Reset, Reset All
  - Vignetting: Correction Manual, Intensity 100
- Bottom Bar:** Nik Collection, Export, and other standard file operations.

# Deshacer-repetir

Puede proporcionar un mecanismo de deshacer y rehacer si las acciones saben cómo revertir su ejecución.

Para ello, cada clase ConcreteCommand puede necesitar guardar información adicional del estado.

# El estado puede incluir

El objeto receptor, quien realmente lleva a cabo las operaciones en respuesta a la petición.

Los parámetros de la operación llevada a cabo en el receptor.

Cualquier valor original del receptor que pueda cambiar como resultado de ejecutar la petición.

El receptor debe proporcionar operaciones que permitan al command devolver el receptor a su estado anterior.

Es posible que haya que copiar las acciones que se pueden deshacer antes de guardarlas en el historial.

Para distinguir entre distintas invocaciones de la misma acción si el estado puede variar entre ellas.

Dichos commands hacen también las veces de prototipos.

DxO PhotoLab – Command – Gilbo.tif (M)

Compare 1:1 62 % Reset Presets

HISTOGRAM

RGB R G B L

219, 220, 220

MOVE/ZOOM

Move/Zoom

Advanced History

Applied Default Preset

Clear History

PRESET EDITOR

Command

Gilbo.tif

1/1 images

Search for corrections

Exposure Correction Manual Exposure 0,00

DxO Smart Lighting

Uniform Spot Weighted

Mode Slight Intensity 25

Selective Tone

Highlights 0 Midtones 0 Shadows 0 Blacks 0

DxO ClearView Plus

Intensity 50

Contrast

Contrast 0 Microcontrast 0

Tone Curve

Curve Master Reset Reset All

Vignetting

Correction Manual Intensity 100

Nik Collection Export

The screenshot displays the DxO PhotoLab software interface with a landscape photograph of a mountainous region. The main workspace shows a wide-angle view of rugged peaks and a river. On the left, there are several panels: 'HISTOGRAM' showing the color distribution; 'MOVE/ZOOM' with a preview window; 'HISTORY' showing the applied default preset; and 'PRESET EDITOR' with various correction tools like Exposure, DxO Smart Lighting, and Tone Curve. The top bar includes standard photo editing tools like crop, rotate, and selection, along with zoom levels (1:1, 62%) and a search bar for corrections.

# Ejemplo de código

# Ejemplo en Swing

<https://alvinalexander.com/java/java-action-abstractaction-actionlistener/>



```
public void actionPerformed(ActionEvent e) {  
    Object o = e.getSource();  
    if (o = fileNewItem)  
        doFileNewAction();  
    else if (o = fileOpenMenuItem)  
        doFileOpenAction();  
    else if (o = fileOpenRecentMenuItem)  
        doFileOpenRecentAction();  
    else if (o = fileSaveMenuItem)  
        doFileSaveAction();  
    // and more ...  
}
```

Sin usar Command (mal)



## Con commands (bien)

```
public void actionPerformed(ActionEvent e) {  
    // create our actions  
    cutAction = new CutAction("Cut", cutIcon, "Cut stuff onto the clipboard",  
        new Integer(KeyEvent.VK_CUT));  
    copyAction = new CopyAction("Copy", copyIcon, "Copy stuff to the clipboard",  
        new Integer(KeyEvent.VK_COPY));  
    pasteAction = new PasteAction("Paste", pasteIcon,  
        "Paste whatever is on the clipboard",  
        new Integer(KeyEvent.VK_PASTE));  
  
    // create our main menu  
    JMenu fileMenu = new JMenu("File");  
    JMenu editMenu = new JMenu("Edit");  
  
    // create our menu items, using the same actions the toolbar buttons use  
    JMenuItem cutMenuItem = new JMenuItem(cutAction);  
    JMenuItem copyMenuItem = new JMenuItem(copyAction);  
    JMenuItem pasteMenuItem = new JMenuItem(pasteAction);  
  
    // add the menu items to the Edit menu  
    editMenu.add(cutMenuItem);  
    editMenu.add(copyMenuItem);  
    editMenu.add(pasteMenuItem);  
}
```

# Patrones relacionados

# Composite

Se puede usar un patrón Composite junto con el Command para implementar macros o transacciones.

# Prototype

Un command que haya que copiar antes de guardarlo en el historial es también un Prototype.

# Recordatorio final

*La esencia del patrón es el...*

*Desacoplamiento  
temporal*



```
public void operation() {  
    Command aCommand = new ConcreteCommand(aReceiver);  
    // ...  
    aCommand.execute();  
}
```

¿Qué os parece? ¿Tiene algún sentido?





Andry Strong

¿Qué sentido tendría crear toda la infraestructura para mover una mera llamada a un método del receptor (pues eso, en esencia, es un Command) a un objeto aparte, si este no se guarda en ningún sitio para poder ser ejecutado más tarde?

Si se crea, se ejecuta y se destruye a continuación, para ese viaje no hacían falta tantas alforjas: no ganamos nada.

Se podría haber dejado el código anterior como sigue y sería lo mismo (solo que muchísimo más sencillo).



```
public void operation() {  
    Command aCommand = new ConcreteCommand(aReceiver);  
    // ...  
    aCommand.execute();  
    // ...  
}
```



```
public void operation() {  
    // ...  
    aReceiver.action();  
    // ...  
}
```