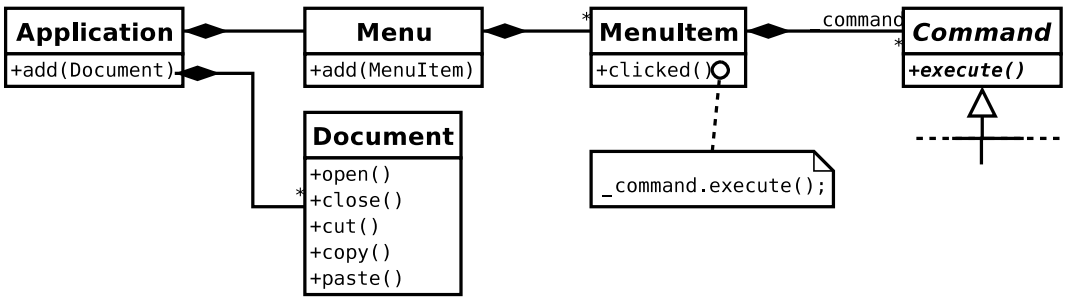
Comando (Command)

- *Patrón de Comportamiento*
- *Propósito*

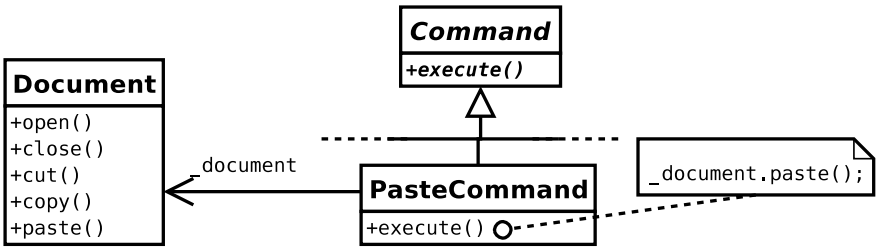
  Encapsula una petición como un objeto

  Permite la parametrización de clientes con diferentes peticiones, crear colas de peticiones y soporte de operaciones cancelables (*undo*)

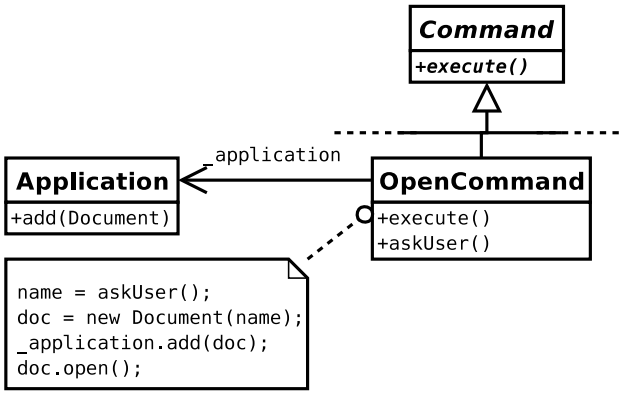
- *Motivación*
  - En ocasiones es necesario realizar peticiones sin:
    - Conocer información acerca de la operación a realizar
    - Conocer información acerca del receptor de la petición
  - Ejemplo: Herramientas para la creación de interfaces de usuario
    - Botones y opciones de menú ejecutan una petición, pero esa petición depende de la aplicación
  - Solución: Convertir la petición en un objeto
  - Clave: Clase abstracta que declara la interfaz para la ejecución de operaciones



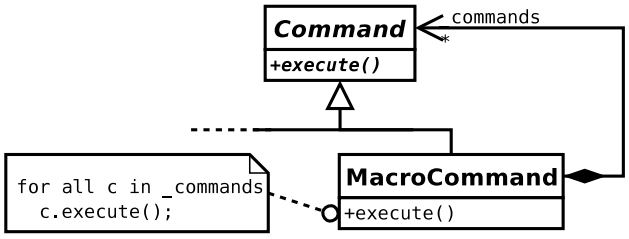
- *Motivación* (cont.)
  - Las subclases concretas de *Command* especifican el par receptor-acción:
    - Atributo que almacena referencia al receptor
    - Comportamiento de la clase define la acción
  - Comando Pegar



  - Comando Abrir

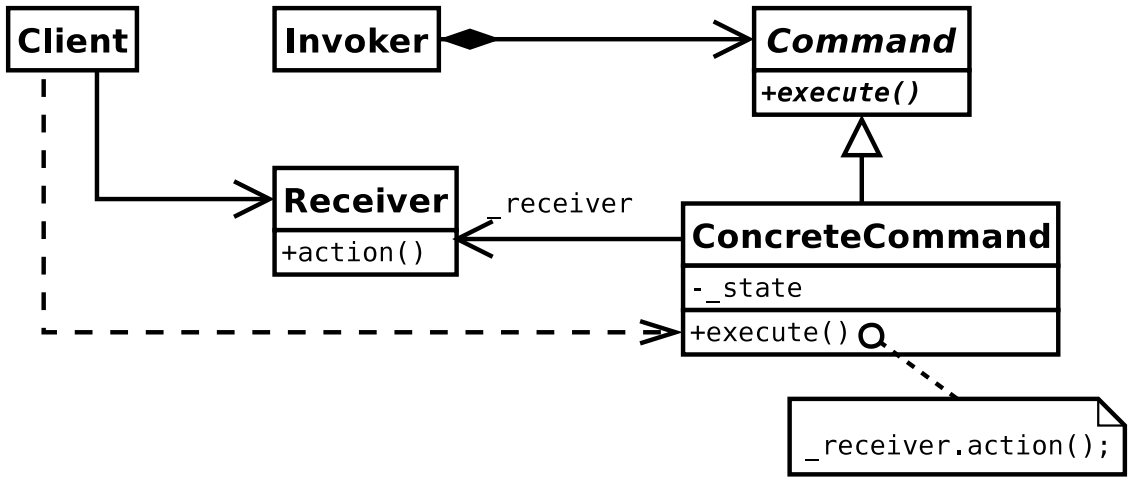


  - Macro-comandos: composición de comandos!



- *Motivación* (cont. II)
  - Fundamental: Separación del objeto que desencadena la operación del objeto que conoce como llevarla a cabo
    - Compartición de comandos por diferentes elementos
    - Cambio dinámico de comandos (sensible al contexto)
    - Composición de comandos (*scripting*)
- *Aplicabilidad*
  - Establecer como parámetro de un objeto la acción a realizar (visión OO de un *callback*)
  - Especificar, encolar, y ejecutar peticiones en diferentes momentos
  - Soporte de *Deshacer*
    - El comando almacena estado para invertir su efecto
    - Añadir método *Unexecute* a la interfaz del comando
    - Comandos ejecutados se guardan en una lista histórica
    - Undo-Redo $\rightarrow$ recorrer la lista adelante y atrás llamando a *Unexecute* y *Execute* respectivamente
  - Soporte de *Logging*
    - Aumentando la interfaz del comando con operaciones de salvar-guardar (*log* persistente de cambios)
    - Recuperación: cargar los comandos guardados e invocar su *Execute* en secuencia
  - Soporte de *Transacciones*

- *Estructura*



- *Participantes*

  - Comando (Command)

    Declara la interfaz para ejecutar una operación

  - Comando Concreto (ConcreteCommand)

    Define un vínculo entre un objeto receptor y una acción

    Implementa la interfaz Comando, invocando la operación sobre el receptor

  - Cliente (Client)

    Crea un Comando Concreto y establece su receptor

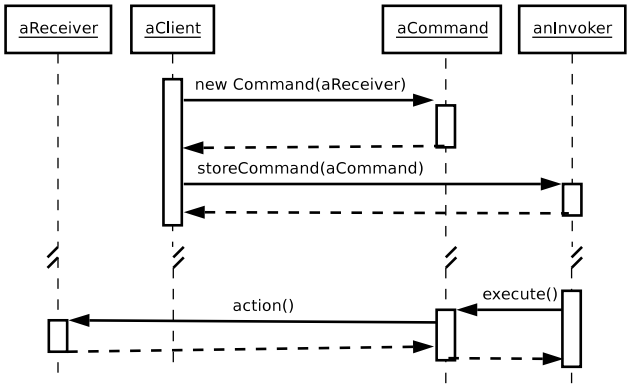
  - Invocador (Invoker)

    Solicita al comando que ejecute su acción

  - Receptor (Receiver)

    Conoce cómo realizar las operaciones asociadas con la ejecución de un comando

- *Colaboraciones*
  - Cliente crea Comando Concreto y especifica su receptor
  - El invocador envía el mensaje *Execute* a un comando
  - El Comando Concreto almacena el estado necesario para deshacer
  - El Comando Concreto invoca las operaciones sobre el receptor para realizar la petición



- *Consecuencias*
  - Desacoplamiento del objeto que invoca la operación del objeto que conoce como llevarla a cabo
  - Comandos pueden ser manipulados y extendidos como cualquier objeto
  - Composición de comandos
  - Fácil añadir comandos: no es necesario modificar clases existentes
- *Implementación*
  - Responsabilidad asignada al comando
  - Soporte de Undo/Redo
  - Evitar acumulación de errores en el proceso de deshacer