ANEXO E: PATRONES DE DISEÑO COMMAND Y COMPOSITE

Un patrón de diseño nos indica una solución estándar para un problema común de programación, es decir nos dan una solución ya probada y documentada a problemas de desarrollo sujetos a contextos similares. En un patrón hay que tener en cuenta elementos como su nombre, clasificación, cuándo deben ser aplicados, la solución abstracta del problema que presentan y sus consecuencias (costos y beneficios).

Una clasificación de patrones de diseño sería:

- Patrones Creacionales: Inicialización y configuración de objetos.
- Patrones Estructurales: Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes.
- Patrones de Comportamiento: Describen la comunicación entre clases.

En particular en la implementación de la aplicación se han utilizado los patrones de diseño *Command* y *Composite*, que se discuten a continuación.

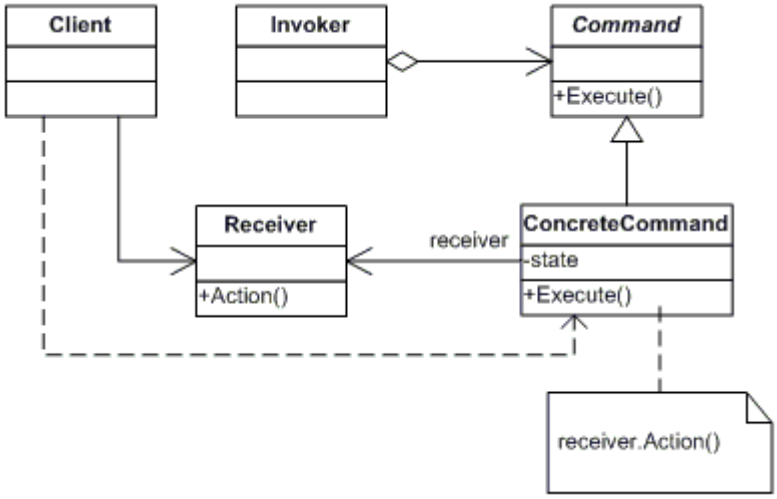
Patrón Command

Este patrón *Command* es un patrón de comportamiento que encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer las operaciones.

- **Nombre**: Command (Comando).
- **Clasificación del Patron:** Patrón de Comportamiento.
- **Intención:** Encapsular un comando en un objeto de tal forma que pueda ser almacenado, pasado a métodos y devuelto igual que cualquier otro objeto.
- **Motivación:** A veces se quiere poder enviar solicitudes a objetos sin conocer exactamente la operación solicitada ni del receptor de la solicitud.

  Por ejemplo un objeto botón o menú ejecuta solicitudes pero la solicitud no está implementada dentro del mismo. Una biblioteca de clases para interfaces de usuario tendrá objetos como botones y elementos de menú responsables de realizar alguna operación en respuesta a una entrada del usuario.

Estructura:



5-1 Patrón Command

Participantes:

- **Command:** Declara la interface para la ejecución de la operación
- **ConcreteCommand:** Define la relación entre el objeto Receiver y una acción, implementa Execute() al invocar las operaciones correspondientes en Receiver.
- **Client:** Crea un objeto ConcreteCommand y lo relaciona con su Receiver.
- **Invoker:** Realiza solicitudes al objeto Command.
- **Receiver:** Sabe cómo ejecutar las operaciones asociadas a la solicitud. Cualquier clase puede ser receptora.

Consecuencias:

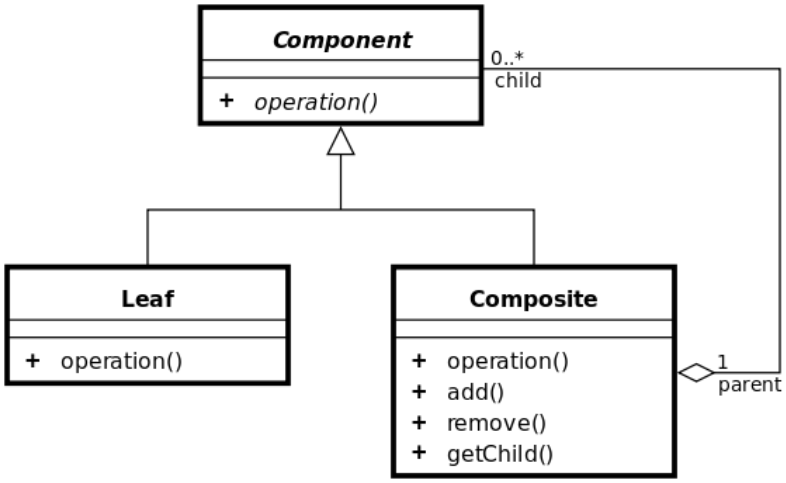
- Command desliga el objeto invocador del objeto receptor
- Los comandos son objetos de primera clase.
- Se pueden ensamblar comandos en comandos compuestos.
- Para un nuevo comando no se necesita extender las clases.

Patrón Composite

Utilizado para construir objetos complejos a partir de otros más simples, utilizando para ello la composición recursiva y una estructura de árbol. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

- **Nombre del patrón:** Composite (Compuesto)
- **Clasificación del patrón**: Estructural
- **Intención:** Componer objetos en jerarquías parte-todo y permitir a los clientes tratar objetos simples y compuestos de modo uniforme.
- **Motivación:** Cuando podemos definir elementos complejos en base a otros más simples. El comportamiento y/o la información que proporciona un elemento complejo está determinada por los elementos que lo componen.

Estructura:



5-2 Patrón Composite

Participantes:

- **Component:** Declara la interfaz para los objetos de la composición, es la interfaz de acceso y manipulación de los componentes hijo e implementa algunos comportamientos por defecto.
- **Client:** Manipula objetos a través de la interfaz proporcionada por Component.
- **Composite:** Define el comportamiento de los componentes compuestos, almacena a los hijos e implementa las operaciones de manejo de los componentes.
- **Leaf:** Definen comportamientos para objetos primitivos del objeto compuesto.