Intención del patrón

- Desacoplar una abstracción de su implementación de manera que ambas puedan variar independientemente.
- Publicar la interfaz en una jerarquía de herencias y ocultar la implementación en su propia jerarquía de herencia.
- Además de encapsulación, se usa para aislamiento.

Se debe usar el patrón de diseño *Bridge* cuando:

- Se quiere un enlace de la implementación en tiempo de ejecución.
- Se posee una proliferación de clases resultado del acoplamiento de la interfaz y numerosas implementaciones.
- Se desea compartir una implementación entre múltiples objetos.
- Se necesita asignar jerarquías de clase ortogonales.

Consecuencias:

- Desacoplar la interfaces del objeto.
- Extensibilidad mejorada. Se puede extender (por ej. Subclase) las jerarquías de abstracción y de implementación independientemente.
- Ocultar detalles a los clientes.

Bridge es un sinónimo para el patrón "*handle/body*". Este es un mecanismo de diseño que encapsula una clase de implementación dentro de una clase de interfaz. El primero es el *body* y el segundo es el *handle*. El *handle* es visto por el cliente como la clase real, pero el trabajo es llevado a cabo en el *body*. Este patrón puede ser usado para descomponer una abstracción compleja en clases más pequeñas y manejables. También puede ser usado para compartir un único recurso entre múltiples clases que lo controlan.

CODIGO

- El patrón *Adapter* hace que las cosas funcionen luego de haber sido diseñadas. por otro lado el patrón *Bridge* hace que funcionen antes del diseño.
- El patrón *Bridge* está diseñado por adelantado para que la abstracción y la implementación varíen independientemente mientras que *Adapter* es retroadaptado para que las clases puedan trabajar en forma conjunta.
- Los patrones *State*, *Strategy*, *Bridge* (y hasta cierto grado el *Adapter*) tienen una estructura similar de solución. Todos ellos comparten elementos del patrón "*handle/body*". Pero se diferencian en la intención, es decir, resolver problemas diferentes.
- La estructura de los patrones *State* y *Bridge*, son idénticas (Con la excepción que *Bridge* admite jerarquías de clase de las clases *handle*, mientras que el patrón de diseño *State* admite sólo una). Los dos patrones usan la misma estructura para resolver distintos problemas: *State* le permite a un objeto cambiar su comportamiento junto con su estado, mientras que la intención del patrón *Bridge* es desacoplar una abstracción de su implementación para que ambas puedan variar independientemente.
- Si las clases de interfaz delegan su creación de sus clases de implementación (en lugar de crearse/acoplarse a ellas mismas directamente) entonces el diseño por lo general usa el patrón de diseño Abstract Factory para crear los objetos de implementación.