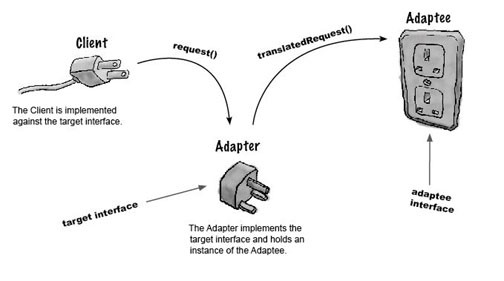
**Patrón Adapter (Adaptador)**

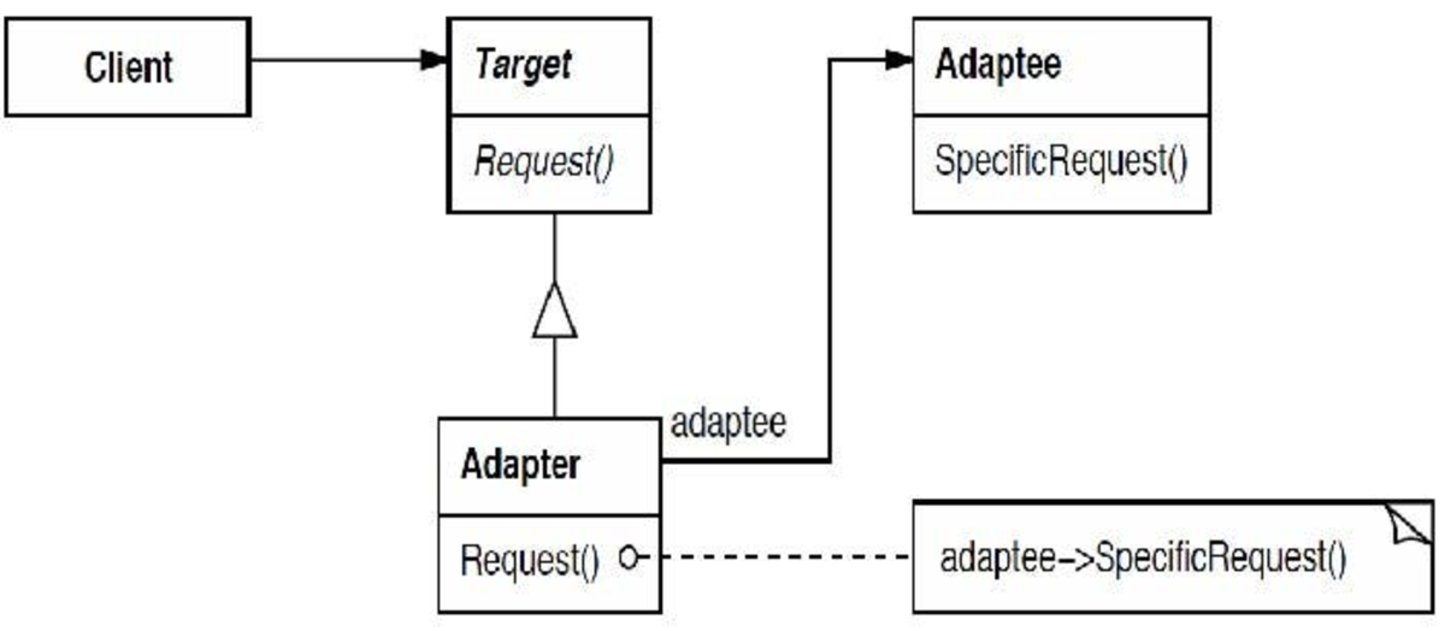
El patrón adapter o patrón adaptador tiene como propósito poder compartir interfaces de una clase en otra interfaz que es la que esperan los clientes. Permite que cooperen las clases que de otra forma no podrían por tener interfaces incompatibles. Este patrón también es conocido como “envoltorio”.

Hay que recalcar que en este patrón habrá un “cliente”, un “adaptador” y un “adaptado”



**Este patrón se puede aplicar cuando:**

* Se necesita usar una clase existente y su interfaz no concuerda con lo que se necesita
* Se quiere crear una clase reutilizable que coopere con clases no relacionadas o que no han sido previstas
* Es necesario usar varias subclases existentes, pero no resulta práctico adaptar su interfaz heredando cada una de ellas



Target: Es una interfaz, la cual usa el cliente en el cual se quiere reutilizar el componente genérico que tenemos (Adaptee), es allí donde radica el problema que soluciona el patrón adaptador, ya que Target es diferente de la interfaz usada por el componente genérico que tenemos (Adaptee).

Adaptee: Es el componente genérico que tenemos, en el cual tenemos implementada cierta funcionalidad que queremos reutilizar desde nuestro cliente, al ser un componente genérico y pensado en cierta ocasión para algún escenario su interfaz es incompatible con Target, es decir la firma de su método puede retornar un valor diferente y recibir parámetros diferentes, es por esto por lo que son incompatibles.

Adapter: Es nuestro adaptador o también conocido como Wrapper, el cual nos permite convertir en compatibles estas dos interfaces que son incompatibles por naturaleza, y como vemos en el diagrama, Adapter es una clase que hereda de Adaptee y que implementa la interfaz Target, de este modo puede cumplir con el contrato esperado por el cliente y a su vez utilizar la implementación existente en Adaptee.

Cliente: Es la aplicación u otro componente, desde la cual deseamos reutilizar la funcionalidad que tenemos en nuestro Adaptee, es por esto por lo que anteriormente nos referimos mucho al cliente.

**Consecuencias de la utilización del patrón adapter:**

* Adapta una clase “Adaptee” a Target, pero se refiere a una clase “Adaptee” concreta
* Permite que un “Adapter” redefina parte del comportamiento del “Adaptee”, por ser un “Adapter” una subclase de “Adaptee” (Solo si hay herencia múltiple)
* Introduce un solo objeto y no se necesita un puntero adicional para obtener el objeto adaptado

**Ejemplo:**

* Se esta desarrollando un sistema que transforma en autónomo un vehículo estándar. Para esto se implementa una clase que gestiona el comportamiento del motor y su integración con el mundo. Funciona a la perfección en motores a explosión. El problema surge con las nuevas tecnologías de motores eléctricos.
* El motor eléctrico deberá estar conectado para poder activarse y activo para poder moverse.

Diagrama

Descripción generada automáticamente