Adapter

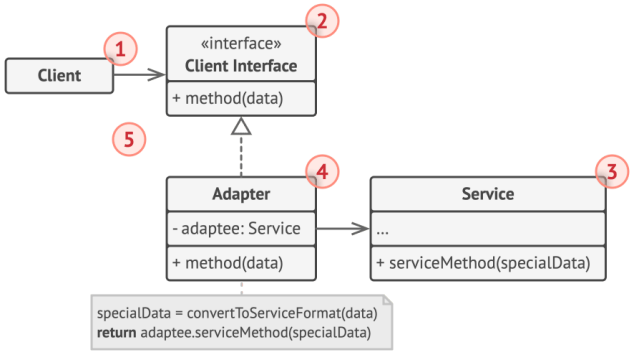
L’adapter è un pattern strutturale che consente la collaborazione di oggetti con interfacce incompatibili.

Si tratta di un oggetto speciale che converte l’interfaccia di un oggetto in modo che un altro oggetto possa comprenderla.

Funzionamento

- Crea un’interfaccia compatibile con un oggetto esistente;
- L’oggetto esistente chiama i metodi dell’adapter;
- L’adapter converte e inoltra le chiamate all’oggetto target nel formato previsto;

Struttura (Adapter oggetto)



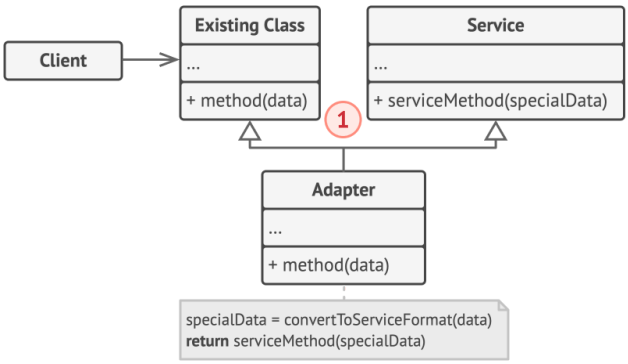
- Il client è una classe che contiene la logica aziendale esistente del programma;
- L’interfaccia client descrive un protocollo che le altre classi devono seguire per poter collaborare con il codice client;
- Il servizio è una classe utile. Il client non può usare questa classe direttamente perchè ha un’interfaccia incompatibile;
- L’adapter è una classe in grado di interagire sia con il client che con il servizio: implementa l’interfaccia client, mentre esegue il wrapping dell’oggetto service.

Riceve le chiamate dal client tramite l'interfaccia client e le traduce in chiamate all'oggetto servizio incapsulato in un formato comprensibile;

- Il codice client non viene accoppiato alla classe adapter concreta, purchè interagisca con l'adapter tramite l'interfaccia client.

Struttura (Adapter di classe)

Questa implementazione usa l'ereditarietà: l'adapter eredita le interfacce da entrambi gli oggetti contemporaneamente.



Tale approccio può essere implementato solo in linguaggi di programmazione che supportano l'ereditarietà multipla (C++).

Applicabilità

- Usare l'adapter quando si vuole usare una classe esistente, ma la sua interfaccia non è compatibile con il resto del codice;
- Usare il modello quando si desidera riusare diverse sottoclassi esistenti prive di alcune funzionalità comuni che non possono essere aggiunte alla superclasse;

Vantaggi

- Principio di singola responsabilità;

- Principio Open/Closed: è possibile introdurre nuovi Adapter senza compromettere il codice client esistente, purchè funzionino con gli adapter tramite l'interfaccia client.

Svantaggi

- La complessità complessiva del codice aumenta perchè è necessario introdurre un set di nuove interfacce e classi.

Relazioni con **altri modelli**

- Bridge viene solitamente progettato in anticipo, consentendo di sviluppare parti di un'applicazione indipendentemente l'una dall'altra. Adapter viene comunemente usato con un'app esistente per far funzionare insieme classi altrimenti incompatibili;
- Adapter fornisce un'interfaccia completamente diversa per accedere a un oggetto esistente. Decorator fornisce un'interfaccia invariata o viene estesa e supporta la composizione ricorsiva cosa non possibile usando adapter;
- Con Adapter si accede a un oggetto esistente tramite un'interfaccia diversa. Con Proxy l'interfaccia rimane la stessa. Con Decorator si accede all'oggetto tramite un'interfaccia migliorata;
- Facade definisce una nuova interfaccia per gli oggetti esistenti, mentre Adapter cerca di rendere usando l'interfaccia esistente. Adapter, solitamente, avvolge un solo oggetto, mentre Facade lavora con un intero sottoinsieme di oggetti;
- Bridge, State, Strategy e in parte anche Adapter hanno una struttura molto simile, si basano sulla composizione, ovvero sulla delega del lavoro ad altri oggetti. Ma risolvono problemi diversi.